

SPECIFICATION

To all whom it may concern:

Be it known that John K. Thomasson and Myron L. Mosbarger, citizens of the United States of America, have invented a new and useful invention entitled METHOD AND SYSTEM FOR ASYMMETRIC SATELLITE COMMUNICATIONS FOR LOCAL AREA NETWORKS of which the following comprises a complete specification.

METHOD AND SYSTEM FOR ASYMMETRIC SATELLITE

COMMUNICATIONS FOR LOCAL AREA NETWORKS

Ins. A₁
Software Appendix. This specification includes a software source code appendix which includes the computer source code of one preferred embodiment of the invention. In other embodiments of the invention, the inventive concept may be implemented in other computer code, in dedicated electronic hardware, in a combination of these, or otherwise. This software appendix is hereby incorporated in this application in its entirety and is to be considered to be part of the disclosure of this specification.

I. BACKGROUND OF THE INVENTION

Ins. A₂
A. Field of the Invention.

This invention relates to methods and systems for communications between computers and other digital information devices. More particularly, this invention relates to communications between computers making use of digital satellite communications channels and computer local area networks, to provide access to the internet, to facilitate data and software distribution, and/or to enhance the capabilities of intranet systems for computers with connections to local area networks.

B. Description of Related Art.

It is well established that computers can communicate across local or wide area networks. It is also well known that satellite receivers and transmitters can be used to transfer high volumes of digital data. Some efforts have been made to provide communication systems which can be

1 used to transfer data between computer processors using a variety of communication mediums
2 (see Moura et al., U.S. Patent No. 5,586,121). However, it is desirable to provide a high-speed,
3 low-cost, satellite-based communication system which is designed to optimize the use of digital
4 satellite systems for local area networks (LANs). Optimizing the use of the digital satellite
5 channel is best accomplished through the use of asymmetrical communications between the
6 computer server and the internet as opposed to symmetric communication, in which substantially
7 the same data rates and the same media are used for both the transmit direction and the receive
8 direction, and as opposed to communication systems which employ asymmetrical communication
9 between the local area network and the server. Particular, asymmetrical systems which require
10 upstream router hardware, "backbone" network hardware, or dial-up internet service providers
11 (ISPs) to create a "hybrid" asymmetrical local system with a symmetrical local area network.
12 Since calls to the internet can efficiently be made at relatively low speeds, and since using digital
13 satellites as a communication medium provides the capability of very high speed responses from
14 the internet, an asymmetric transmission from the internet across the digital satellite to the LAN
15 server provides the greatest system efficiency.

16 The most common method of sending and receiving computer information today is a land
17 line service (i.e., a switched service, a dedicated line, and/or an analog modem, each using
18 telephone wire lines). However, such a system encounters many problems, including slow
19 transmission speeds, high access costs, lack of available wire lines, and internet congestion.

20 Satellite communication receivers are commonly used to create or supplement existing
21 private wide area data and video networks. When used as an extension to a data network, these
22 satellite links may interconnect local area networks. Satellite links can provide many advantages

1 over land line service, including potentially high speed data transmission and wide availability.
2 However, typical satellite links have required expensive hardware both to transmit and to receive
3 data. The expense of the hardware has made the use of satellite communication channels
4 generally unavailable to those who most need it.

5 This invention addresses these issues by providing a method and system for providing the
6 advantages of satellite communications for high volume download data packets and typically
7 using a relatively low speed land line for the low volume upload data request packets. By
8 capitalizing on the asymmetrical nature of internet dataflow, this invention provides an efficient
9 solution for LAN to satellite internet communications.

10 For general background material the reader is directed to U.S. Patent Nos. 5,095,480,
11 5,379,296, 5,423,002, 5,488,412, 5,534,913, 5,539,736, 5,541,911, 5,541,927, 5,555,244,
12 5,583,997, 5,586,121, 5,594,872, 5,610,910, 5,610,920, 5,631,907, 5,659,692, 5,668,857,
13 5,673,265, each of which is hereby incorporated by reference in its entirety for the material
14 disclosed therein.

15 II. SUMMARY OF THE INVENTION

16 This invention is a method and system for efficiently communicating between networked
17 computers using a high speed satellite communications channel. It is an object of this invention
18 to provide a high speed satellite based information delivery system for local area network
19 connectivity to the internet, for file, data, software, and/or multimedia distribution.

20 It is a further object of this invention to provide a data transmission system particularly
21 well suited to remote location and/or locations where access to high speed data mediums is
22 unavailable or prohibitively expensive.

1 It is a further object of this invention to provide a high speed data transmission system
2 that utilizes a highly flexible and adaptable software method.

3 It is a further object of this invention to provide a high speed data transmission system
4 that communicates with the internet while being internet service provider (ISP) independent.

5 It is a further object of this invention to provide a high speed data transmission system
6 that makes use of digital satellite communications technology to enhance data bandwidth,
7 channel reliability, and accessibility.

8 It is a further object of this invention to provide a high speed data transmission system
9 that utilizes a software method capable of operating on a wide range of server operating systems,
10 including Windows 95, Windows NT, NetWare, Linus, Macintosh, present and future versions
11 and the equivalents.

12 It is a still further object of this invention to provide a high speed data transmission
13 system that is compatible with a wide range of communication protocols and/or mediums,
14 including ISDN, T1, modem, dedicated phone line, switched phone line, frame relay and ATM.

15 It is another object of this invention to provide a method for permitting many client
16 computer systems, which may be operating system independent and operating on one or more
17 local area networks, to communicate over a single satellite dish, at very high data rates.

18 It is a further object of this invention to provide a method using software which can
19 operate on a wide variety of hardware, operating system, and software platforms, including, but
20 not limited to: Macintosh, Linux, Unix, OS/2, and Windows NT.

21 These and other objects of this invention are readily apparent to individuals of ordinary
22 skill in the art upon further study of the drawings, detailed description, claims and abstract that

are included in this patent disclosure.

III. BRIEF DESCRIPTION OF THE DRAWINGS

Figure 1 depicts a top level rendering of the major component parts of the communication system invention.

Figure 2 depicts a preferred embodiment of the architecture of the invention.

Figure 3 depicts the preferred flow of data through the several protocol transitions in a preferred embodiment of the invention.

Figure 4 depicts a top level flow diagram showing the primary steps of the process flow for an example single received data packet in the preferred method of the invention.

Figure 5 depicts a detailed flow diagram showing the package delivery major step of the preferred embodiment of the invention.

Figure 6 depicts a detailed flow diagram showing the internet protocol (IP) major step of the preferred embodiment of the invention.

Figure 7 depicts a top level flow diagram showing the primary steps of the process flow for an example transmission of internet protocol datagrams.

Figure 8 depicts additional detail showing the transfer queue (TxQ) thread processing of the filter queue step of the transmission portion of the invention.

Figure 9 depicts additional detail showing new queue (NewQ) thread processing of the filter queue step of the transmission portion of the invention.

Figure 10 provides an example embodiment of the user interface of the invention.

IV. DETAILED DESCRIPTION OF THE INVENTION

This invention is a system and method for asymmetric communications, between a

1 remote information provider and a client computer system residing on a local area network
2 (LAN), using high bandwidth digital satellite communication channels. The preferred
3 embodiment of the method of the invention is performed in software residing on a local
4 computer system. The current preferred embodiment of the method software ^{is} ~~is~~ written in Intel
5 386 assembly code, C and C++ computer languages. The reader is directed to the ~~appended-~~
6 ^{microfiche} computer software appendix for a complete disclosure of the software making up the current best
7 mode of the method of this invention. Alternatively, those of ordinary skill in the art could
8 practice this method in a wide variety of procedures, computer languages, or even in dedicated
9 electronic hardware. Therefore this patent should not be read to be limited to the specific
10 embodiment of the provided software ^{microfiche} ~~appendix~~. Rather, this software source code is provided to
11 fully describe one preferred embodiment of the method of this invention. Also, in its preferred
12 embodiment, this invention performs in association with DirectPC satellite receivers, Novell
13 NetWare network software, and standard off-the-shelf computer hardware. Other alternative
14 satellite receiver systems, networking software and computer hardware could easily be
15 substituted by those of ordinary skill in the art without departing from the essence of this
16 invention. Similarly, the preferred embodiment, described in the following detailed
17 description, includes a number of components and method steps which may not be absolutely
18 necessary in other embodiments of the invention. The reader is, therefore, directed to the claims
19 for a description of the range of this patent.

20 Figure 1 depicts a top level rendering of the major components and communication paths
21 constituting the system 100 of this invention. A client computer 101a-g is shown clustered with
22 other client computers on a local area network (LAN) 102. The client computer 101 uses

1 standard off-the-shelf commercially available software, while the server provides the user
2 interface to the desired information. The LAN 102 provides the means for communicating from
3 the client computer 101 to a server 103, which provides the communication interface outside the
4 LAN. The server 103 receives data from a digital satellite receiver 110, depicted here as a
5 satellite dish, across a signal antenna waveguide 115. The digital satellite receiver 110 receives
6 the digital information from a downlink channel 111, which is transmitted from a
7 geosynchronous satellite 112. The satellite 112 receives the information from a network
8 operations center 114 via an uplink channel 113. The server 103 generally uses the above
9 described communications channel from the network operations center 114 for downloads of
10 information, such as internet web page data, software updates, data file distributions and other
11 similar data packages. Alternatively and in addition, this invention provides the capability of
12 using another satellite communication channel to send requests from the client computer 101 to
13 the network operations center 114. This is a particularly useful feature for access from remote
14 locations. Use of the satellite communication channel provides important benefits to the
15 information requestor at the client computer 101, including very high speed data transfer, the
16 ability to receive broadcast software distributions which in turn means the requestor is likely to
17 receive such distributions in a more timely and cost effective manner, and the ability to have
18 internet access from locations where wired communications channels, such as telephone lines,
19 are either unavailable, overly burdened, or prohibitively expensive.

20 Figure 1 also shows the preferred, and more typical request communications channel. In
21 this preferred request channel the client computer 101, connected through the LAN 102, through
22 the server 103, sends a request via modem 105, which typically is connected to the server via a

1 standard serial RS-232 cable 104. The modem 105 in turn is connected to standard telephone
2 land lines 107 via a standard phone cable 106. The request is passed across the land lines 107 to
3 the internet service provider 108, which communicates to the internet 109.

4 This invention is designed to be highly flexible and adaptable to different client computer
5 101 configurations, both hardware and software as well as with and to a wide variety of
6 communication interfaces. Computer hardware such as personal computers, workstations, mini
7 computers, mainframe computers and special purpose computational equipment can be
8 functional client computers 101 as intended within this patent specification. Similarly, computer
9 system operating systems which are supported and used in the preferred and alternative
10 embodiments of this invention included but are not limited to: Windows 3.1, Windows 95,
11 Windows NT, Macintosh, Linux, Unix, OS/2, NetWare, their current versions, past versions, and
12 equivalent future versions and the equivalent. Communications interfaces that are or can
13 alternatively be used with or as a part of this invention include routers, ethernet, ISDN
14 equipment, switched 56, T1, Token Ring, frame relay, modems, satellite and the equivalent.

15 The advantage of this preferred mode of operation is that the communication channels are
16 used in the most efficient manner. Typically, request packets are relatively small and can be
17 transferred with minimal impact across land lines. While downloaded packets can be very large
18 with significant amounts of highly concentrated graphics. For the vast majority of client
19 computer 101 users the limitation of internet or the ability to receive other downloaded file
20 information is the time it takes for the download transfer to be accomplished. This problem is
21 solved by transferring the potentially very large downloaded packets (files, graphics and other
22 information) using the high bandwidth satellite channel.

Figure 2 depicts a preferred embodiment of the software architecture of the invention. As shown, in its preferred embodiment, this software operates in association with several standard commercially available software packages and protocols, including Novell NetWare, Ethernet, Token Ring, TCP/IP, IPX and AIO. This software method of the invention also makes use of certain commercially available hardware component, as shown in figure 2, including: the satellite receiver 110, a network router 205 and a modem 104. The reader should understand that this figure 2 presents a single simplified embodiment of the invention. Alternative embodiments could use alternative software packages and protocols, as well as different or multiple hardware components. Figure 2 also shows a single embodiment of the path of information through the various software and hardware components. Download information packets are received by the satellite receiver 110, which in turn communicates electronically with the DPC LAN 210, a commercially available hardware driver. The information next passes through the LSL NLM 202, a routine commercially provided by Novell Incorporated which acts as an intermediary between the driver and the protocol stack to control information packet transfer. Next, the TCP/IP 203 protocol stack receives the information packet. The TCP/IP 203 protocol stack is capable of communicating alternatively with a modem 104, via the LSL NLM 202; the DPCAGENT 207 routine, core to this invention; the AIO 208, a Novell product for managing serial communications; and through the AIOCOMX 209, which is the asynchronous input/output interface to the client computer hardware communication ports, or with router 205, via the LSL NLM 202, the DPCAGENT 207, the LSL NLM 202 and an ethernet driver 204. Alternative embodiments of this invention may make use of other standard commercially available communication protocols, drivers, hardware and software.

Figure 3 depicts the flow of information in the preferred embodiment of the invention from the satellite 112 to the client computer 101 as well as the flow of information back to the internet 109 from the client computer 101. Data is received 301 from the satellite 112 by the satellite receiver 110. Next, the data is transmitted to and received by the server 103 hardware 302 where it is placed in on-board memory. The DPC.LAN DirecPC network card driver retrieves the data packet from hardware memory 303. Next, if the packet is identified as an internet protocol (IP) format packet it is delivered to the IP protocol stack 304. If the packet is identified as a transmission control protocol (TCP) segment, it is delivered to the TCP protocol stack 305. TCP delivers the data packet to a proxy gateway 306. The proxy gateway forwards the data packet to the client computer via the local area network and standard LAN protocol controllers 307. Next, the client computer processes the data and generates a return packet 308. The return packet is delivered to the proxy gateway via the local area network and the standard LAN protocol controllers 309. The return packet is forwarded to the TCP stack 310, and next to the IP stack 311. The IP delivers the return packet to the DPCAGENT.NLM process 312, which delivers the return packet data to a transmit device, such as a modem or a router 313.

Figure 4 depicts a top level flow chart rendering of the major steps of the process flow for a single downloaded data packet section of the invention. Initially the downloaded packet is received 410 by the DirecPC circuit card. This circuit card next transfers the packet data to the DPC.LAN routine 402. In the preferred embodiment of the invention, as shown in the source code appendix, the DPC.LAN routine is denoted as DriverISR proc. This process includes the steps of setting up the RAM adapters and establishing a timestamp for the packet. Further detailed information on the functioning of this routine is found within the software appendix.

1 Next, a test is made 403 as to whether the received data is a package delivery or an internet
2 delivery. If the received data is package data, it is delivered 404. Package data delivery includes
3 and provides the capabilities of simultaneously broadcasting software upgrades or data files to
4 many client computers, potentially throughout any one continent. Client computers can also
5 request the package data delivery service to retrieve a package of information through the client
6 accessible interface of the invention. If the received data is internet data, then internet data
7 delivery is made 405. Additional detail on steps 404 and 405 follows in this specification.

8 Figure 5 depicts a detailed flow chart of the preferred embodiment of the package
9 delivery major step of the method of the invention. The data packet is transferred to the
10 DPCAGENT,NLM process 501. This process is a NetWare Loadable Module (NLM) process
11 running on NetWare. After the data packet is received 501, a test is made to determine whether
12 the packet will update the catalog 502. If the catalog will be updated, then it is updated 503
13 using off-the-shelf commercially available software and the process of package delivery for that
14 packet ends 511. If, however, the catalog will not be updated, then a test is performed to
15 determine whether the site will be updated by the data packet 504. Site updates include
16 modification of such site parameters as NOC versioning, encryption key updates, and becoming a
17 member of a group or leaving a group. If the site will be upgraded, then the process performs the
18 upgrade of the site parameters 505, and the process for this packet ends 511. If the site will not
19 be updated, then the package file is found and stored on the server disk 506. A test is then made
20 to determine whether the end-of-file has been encountered 507. If the end-of-file has not been
21 encountered then the process for that packet ends 511. However, if the end-of-file has been
22 encountered, then a test is made to determine whether there are any "holes" in the file, that is

whether the file is incomplete 508. If no holes are found in the file, it is marked as complete 509 and the process for this packet ends 511. If "holes" are found in the file, then a request for partial retransmit of the missing packet is sent 510, at which point the process for this packet ends 511.

Figure 6 depicts a detailed flow chart of the preferred embodiment of the internet protocol delivery major step of the method of the invention. Internet package delivery or Internet Protocol (IP) delivery is a major function of the invention providing the capability of receiving large files from an internet source at a very high speed. First the data packet is transferred to the DPCAGENT.NLM routine 601. A test is made to determine whether the data is in transmission control protocol (TCP) 602. If the data is not in TCP protocol then the data packet is transferred to the Internet Protocol (IP) stack 609 and the process for this data packet ends 610. If the data is in TCP form then a test is made to determine if a "SYN" or beginning of section is being initiated 603. If no "SYN" is detected, then a test is made to determine if an end of session, commonly a FIN or RST command, has been encountered 605. If no such end of session is found, then the data packet is transferred to the IP stack 609 and the process for this packet is ended 610. If, however, a "SYN" is detected, then the inquiry is made as to whether a connection slot is available 604. Connection slots perform the function of managing the number of subscribers permitted to have access to the communication network at a given time. If a connection slot is available, it means that the customer still has client computer access capacity. If a connection slot is available, a connection slot is allocated 607 and the data packet is transferred to the IP stack 609 and the process ends. If it is determined that a connection slot is not available, then the data packet is dropped or discarded 606 and the process for this packet ends 610.

Figure 7 depicts a top level flow chart showing the primary steps of the preferred embodiment of the transmission of internet protocol datagrams (or packets) method steps of the invention. Packets that are received from the IP stack are stored on the NewQ 701. Next the packet is removed from the NewQ and tested against the each and every packet on the TxQ to determine if any TxQ data is redundant or dated and should be replaced 702. If a comparison of the packet with the TxQ packets finds the updated or "newer" information, then the TxQ packet data is replaced by the current packet data. This approach is essential to maintaining the fairness of the TxQ packet transfers while ensuring that good data is transmitted thereby improving the transmission efficiency of the system. A test is performed to determine if the packet was included in the TxQ 703. If the packet was not included then the current or NewQ head packet is dropped or discarded 704. Otherwise, if the packet was included, a test is performed to determine if the NewQ is empty 705. If the NewQ is not empty, the process returns to the test NewQ step where a new NewQ head packet is compared against the TxQ. If, however, the NewQ is empty, then the process enters a wait state 706 where a trigger, that is meeting a specified condition, such as new packet on the NewQ or exit command, is required before the process restarts at the testing step 702.

Figure 8 depicts a detailed flow chart showing the transfer queue (TxQ) thread processing steps of the transmission portion of the invention. In the preferred embodiment of the invention processing the TxQ and processing the NewQ are independent threads of the program which are capable of running independently on one or more computer processors. In processing the TxQ it is first determined whether the TxQ is empty 801. If the TxQ is empty, then the process enters a wait state 805 where a trigger, such as a polling timer, a transmission complete signal, or an exit

1 command, is required to resume processing. Note that in the preferred embodiment of the
2 invention the expected wait time is calculated in this step and the polling time is initiated. If the
3 TxQ is not empty, a test is made to determine if the head packet of TxQ is too old 802. In the
4 preferred embodiment of the invention, too old is defined as a packet that has been in the TxQ
5 for more than sixty (60) seconds. Alternative embodiments could employ any practicable time
6 period. If the TxQ head packet is too old, then it is discarded 806 and the process returns to the
7 TxQ empty test 801. If the TxQ head packet is not too old, then a test is made to determine if the
8 media, or communication conduit, is capable of transferring another packet of data 803. If the
9 media is capable of transferring another packet, then the packet is written to the transmission
10 device 804, otherwise, the process enters the wait state 805 and waits for a trigger as described
11 above.

12 Figure 9 depicts a detailed flow chart showing the new queue (NewQ) thread processing
13 steps of the filter queue step of the transmission portion of the invention. The filter queue
14 processing step of the invention, which is the core of step 702, is important in providing the
15 communication efficiency which is one of the key objectives of this invention. A test is made to
16 determine whether the NewQ is empty 901. If the NewQ is empty then a wait state is entered
17 902 where a trigger, such as new packet available in NewQ, exit or timer count, is required to
18 resume the process. If NewQ is not empty, then the head of NewQ is renamed as ECB 903, a
19 packet holding variable. The maximum age of the ECB packet is set 904 and a test is performed
20 to determine if the ECB packet is fragmented 905, that is whether ECB is only a partial packet,
21 which in the current best mode of the invention is not inspected. If ECB is fragmented, then it is
22 appended 906 to the TxQ for transmission. If ECB is not fragmented, then a test is performed to

determine if it is a TCP packet 907. If it is not a TCP packet, then the test is made to determine if it is a UDP packet 980. If it is not a UDP packet or a TCP packet then it is appended to the TxQ for transmission 906. If, however, it is a UDP packet, a test is made to determine if ECB is a duplicate of a DNS request 909. If ECB is not a duplicate of a DNS request, then the packet is appended to TxQ 906. If ECB is a duplicate of the DNS request, then it is discarded 910 and the process returns to testing to see if NewQ is empty 901. If it is a TCP packet, then a test is made to determine if ECB is a "SYN" or beginning of session packet 911. If ECB is a "SYN" packet, then a test of whether a connection slot is available is made 912. If a connection slot is available it is allocated 913. If, however, a connection slot is unavailable ECB is discarded 910. If ECB is not a "SYN" packet then the test is made to determine if ECB is a session abort packet (RST) 914. If ECB is found to be a session abort packet TxQ is cleared of all matching packets 915 and a test is made to determine if ECB will terminate a session 916. If ECB is not a session abort then the step of clearing all matching packets 915 is skipped. If the ECB is found to terminate a session, then the a connection slot is released 917. A test is then performed to determine if ECB is void of user data 918. If it contains user data then it is appended to TxQ 906. Otherwise, the End-of-Window-Position (EOWP) is computed 919 and a test is made to determine if ECB's EOWP is greater than the matched TxQ packets 920, if not ECB is discarded 910, and if so, ECB has its EOWP stored in matched TxQ 921 and then ECB is discarded 910.

^{a-k}
Figures 10a~~a~~ show various screen shots demonstrating the user interface of the invention.

Figure 10a shows the DCPAGENT routine options for user selection as well as the digital package delivery queue screen. Figure 10b shows a package of data in transit as displayed on the user screen. Figure 10c shows the main screen of the package delivery interface. Figure 10d

1 shows package statistics as displayed for user information. Figure 10e shows configuration
2 control screens where the user can modify certain modem, package delivery and provider
3 configuration information. Figure 10f shows the package delivery configuration editor screens
4 with the information that can be user modified. Figure 10g shows the login script editor and the
5 provider configuration editor. Figure 10h shows additional provider configuration editor screens
6 showing the configuration of an outbound protocol case. Figure 10i shows the dish or antenna
7 pointing adjustments screens. Figure 10j shows the satellite dish signal strength meter for dish
8 alignment. Figure 10k shows the adapter information screen, here showing site information
9 including the card hardware serial number, the site identification, the status of keys, and a list of
10 communities or groups in which the user is participating.

COMPUTER SOFTWARE APPENDIX

The following computer software appendix is being deposited as part of the specification of this patent application under 37 C.F.R. 1.96 as original copies from computer printout and as two sheets per one patent specification page for the ease of the publication office.

```

#include "dpcagent.h" /* Our header file */

/*****
 * History:
 * Version 0.1 - First Package delivery for demo use.
 * Version 0.2 - (03-15-96)
 * Site ID filtering to catalog
 * Delivery support
 * Stats()
 * sig into file and added DPCUpdateConfig()
 * Version 0.3 - (03-25-96)
 * Support for interactivity field in
 * ee... in catalog by adding minimum
 * ore committing entry.
 * Version 0.4 - (03-29-96)
 * indexes to modem configuration and
 * o 2400 to fix key reception bugs.
 * to enable agent to continue getting
 * useful, even if agent shuts down.
 * Version 0.5 - (04-09-96)
 * showing up in Cancel Download list
 * statistics halfway running(doesn't abend)
 * explicit) files in catalogue
 * ueast/response/confirm code running but
 * it yet.
 * Version 0.6 - (04-10-96)
 * nting Get MLID stats
 * aStrength()
 * Version 0.7 - (04-11-96)
 * igation Editor
 * ength Meter
 * Version 0.8 - (04-12-96)
 * aStrength to report 0 when not connected
 * nting cost(explicit) file download
 * View Database Entries option worked
 * entire entry and is more usable.
 * Version 0.9 - (04-16-96)
 *****/

Added Community/
Added Periodic d
Added DPCGetMLID
Broke modem conf
Added SFX_PUSH s
catalog.
Fixed FreeFreeFr
length check bef
Added baud rate
defaulted them t
Also added code
Keys until succe
Fixed push files
Got DisplayMLIDS
Recognized cost(
Got explicit req
NOC doesn't send
Finished impleme
Added DPCGetSign
Added Modem Conf
Added Signal Str
Fixed DPCGetSign
Finished impleme
Modified the way
so that it shows
Completed Turbo

Internet support. Can't totally test
away is up and running.
* Version 0.10 - (04-24-96)
ck
lse" in IPSendroutine after call
ts. Turbo Internet works 90%.
* Version 0.11 - (04-25-96)
AppendStringField in DLO to include
instead of NULL.
* Version 0.12 - (05-16-96)
ase 2.
between ATDT and 1-800...
r commas in phone numbers and prefix
ing to accept & and -
Gateway strings to have (ISP) follow it
played on if -DEBUG is on command line
* Version 1.00 - (06-11-96)
ud rate to Modem Status string
it doesn't stay in during Internet traffic
formation screen
er edit strings to allow alphas
buffer to 4K(default was 1K)
* Version 1.01 - (06-17-96)
iver comm code into driverio.c
ster call to DPC.LAN so that it could
removes which would allow us to prevent
* Version 1.02 - (06-25-96)
ch to disable package delivery for debug
* Version 1.04 - (06-25-96)
time to send directly to AIO.
* Version 1.05 - (06-28-96)
umber in menu to decimal
size field to modem configuration
which changed configuration screens
* Version 1.06 - (07-05-96)
te tinet fragment routines(jkt).
rol menu.

yet until IJ gat
Added ARP loopba
Fixed missing "e
to GatherFragmen
Fixed MWSAppend
character sets 1
Completion of Ph
Places T in
Added subrout fo
Allowed Init str
Modified IP and
Debug screen dis
Added connect ba
Fixed TX LED so
Added Adapter In
Fixed phone numb
Bumped AIO write
Broke out LAN dr
Added Agent Regl
call us when it
abends.
Added -NOPD swit
Revised send rou
Changed serial n
Added AIO buffer
Added auto login
Completely rewro
Added Modem Cont
Added GNU condit

```

```

lonels.
pt timeouts
utimes into ppp.c
*****
#define AGENT_VERSION INMSG('1.20 ', 173)
/*****
* Global variable used by DPCAGENT.C
*****
*/
/* Resource Tag variables.
*/
struct ResourceTagStructure *NLCHandle = 0;
struct ResourceTagStructure *allortag = 0;
struct ResourceTagStructure *timerTag = 0;
struct ResourceTagStructure *AETag = 0;
struct ResourceTagStructure *asynctag = 0;
/*
* NUT and screen ID variables.
*/
NUTInfo *NUTHandle = NULL;
NUT handle /* main
*/
/* if DEBUG_ALL
struct ScreenStruct *DebugScreenID = 0; /* for OutputToScreen
*/
#endif
/* Screen Height(25)
*/
/* Screen Width(80)
*/
/* Background Portal
*/
/* Process and thread variables.
*/
BYTE
inter to messages /* OS Po
LONG
/* Main Handler thread PID
*/
LONG
/* Packet Handler thread PID
*/
LONG
/* Modem Handler thread PID
*/
LONG
/* Access thread PID
*/
LONG
/* Turbo Internet thread PID
*/
BYTE
/* All threads must exit
*/
/* Tinet needs to wake up flag
*/
/*
*/
/* CRC table used by calcerc().
*/
static unsigned short crcTab[256] = {
0x0000, 0x1189, 0x2312, 0x329b, 0x4624, 0x57ad, 0x6536, 0x74bf,
0x8c48, 0x9dc1, 0xae5a, 0xbde5, 0xc97e, 0xf8f7,
0x1081, 0x0108, 0x3393, 0x221a, 0x56a5, 0x472c, 0x75b7, 0x643e,
0x9c69, 0x8d40, 0xbfdb, 0xae52, 0xdaed, 0xcb64, 0xe19f, 0x6876,
0x2102, 0x308b, 0x0210, 0x1399, 0x6726, 0x76af, 0x4434, 0x55bd,
0xad4a, 0xbcc3, 0x8e58, 0x9fd1, 0xeb5e, 0xf6e7, 0xc87c, 0x83f5,
0x3183, 0x200a, 0x1291, 0x0318, 0x77a7, 0x662e, 0x54b5, 0x453c,
0xbdc6, 0xac42, 0x9e49, 0x8f50, 0xfbee, 0xea66, 0xd8f3, 0xc974,
0x4204, 0x538d, 0x6116, 0x709f, 0x0420, 0x15a9, 0x2732, 0x36bb,
0xc64c, 0xdfc5, 0xed5e, 0xfc67, 0x8868, 0x9961, 0xab7a, 0xbaf3,
0x5285, 0x430c, 0x7197, 0x601e, 0x14a1, 0x0528, 0x37b3, 0x263a,
0xdec6, 0xcfc4, 0xfdd4, 0xec56, 0x98e9, 0x8960, 0xbbfb, 0xaa72,
0x6306, 0x728f, 0x4014, 0x519d, 0x2522, 0x34ab, 0x0630, 0x17b9,
0xef4e, 0xfec7, 0xc5c5, 0xdd45, 0xa96a, 0xb8e3, 0x8a78, 0x99f1,
0x7387, 0x620e, 0x5095, 0x411c, 0x35a3, 0x242a, 0x16b1, 0x0738,
0xfefc, 0xee46, 0xdbc6, 0xcd54, 0xb96b, 0xa862, 0x9a69, 0x8b70,
0x8408, 0x9581, 0xa71a, 0xb693, 0xc32c, 0xd3a5, 0xe13e, 0xf0b7,
0x0840, 0x19c9, 0x2b52, 0x3adb, 0x4e64, 0x5fed, 0x6f76, 0x7efc,
0x9489, 0x8500, 0xb79b, 0xa612, 0x9a62, 0xc324, 0xf1bf, 0xe036,
0x18c1, 0x0948, 0x3bd3, 0x2a5a, 0x5e65, 0x4f6c, 0x7df7, 0x6e7e,
0xa50a, 0xb483, 0x8618, 0x9791, 0x632e, 0x72a7, 0xc03c, 0xd1b5,
0x2942, 0x38cb, 0x0a50, 0x1bd9, 0x6f66, 0x7e6f, 0x4c74, 0x5d4d,
0xb58b, 0xa402, 0x9699, 0x8710, 0xf3af, 0xe226, 0xd0bd, 0xc134,
0x39c3, 0x284a, 0x1a61, 0x0b58, 0x7fe7, 0x6e6e, 0x5c65, 0x4d7c,
0xc60c, 0xd785, 0xe51e, 0xf497, 0x8028, 0x91a1, 0xa33a, 0xb2b3,
0x4a44, 0x5bcd, 0x6956, 0x78d4, 0x0c60, 0x1d69, 0x2f72, 0x3efb,
0xd68d, 0xc704, 0xf59f, 0xe416, 0x90a9, 0x8120, 0xb5bb, 0xa432,
0x5ac5, 0x4b4c, 0x79d7, 0x685e, 0x1c61, 0x0d68, 0x3f63, 0x2e7a,
0xe70e, 0xf687, 0xc41c, 0xd595, 0xa12a, 0xb0a3, 0x8238, 0x93b1,
0x6b46, 0x7ac6, 0x4854, 0x59d4, 0x2d62, 0x3c63, 0x0e70, 0x1fef,
0xf78f, 0xe606, 0xd49d, 0xc514, 0xb1ab, 0xa022, 0x92b9, 0x8330,
0x7bc7, 0x6a4e, 0x58d5, 0x495c, 0x3de3, 0x2c6a, 0x1ef1, 0x0f78
};
/* MLID Statistic globals */
int
GenericDescriptionTable[22*8] =
{
INMSG("Total packets sent:", 103),
INMSG("Total packets received:", 141),
INMSG("No ECB available count:", 148),
INMSG("Send packet too big count:", 156),
INMSG("Reserved:", 186),
INMSG("Receive packet overflow count:", 214),
INMSG("Receive packet too big count:", 215),
INMSG("Receive packet too small count:", 216),
INMSG("Send packet miscellaneous errors:", 217),
INMSG("Receive packet miscellaneous errors:", 218),
INMSG("Send packet retry count:", 223),
INMSG("Checksum errors:", 290),
INMSG("Hardware receive mismatch count:", 291),
INMSG("Total send OK byte count low:", 229),
INMSG("Total send OK byte count high:", 230),
INMSG("Total receive OK byte count low:", 231),
INMSG("Total receive OK byte count high:", 232),
INMSG("Total group address send count:", 233),
INMSG("Total group address receive count:", 251),
INMSG("Adapter reset count:", 252),
INMSG("Adapter operating time stamp:", 253),
INMSG("Adapter queue depth:", 255),
INMSG("Send OK single collision count:", 159),
INMSG("Send OK multiple collision count:", 256),
INMSG("Send OK but deferred", 264),
INMSG("Send abort from late collision", 265),
INMSG("Send abort from excess collision", 266),
INMSG("Send abort from carrier sense", 267),
INMSG("Send abort from excessive deferral", 268),
INMSG("Receive abort from bad frame alignment", 269)
}

```

```

);
#define STATS_DATA_WIDTH 13
#define FSD_DATA_WIDTH 40
#define BORDER_WIDTH 3
#define INDENT_WIDTH 3

void ( *BackgroundFuncPtr ) ( LONG portalNumber ) = NULL;
LONG BackgroundPortal;
int GenericLineStart;
int GblDataCol;
struct DOSCountryInfoStruct GblDOSCountryInfo;

int DebugFlag = FALSE;

/* The array nDeclination contains the Declination in degrees to add or
 * subtract from true azimuth to get magnetic azimuth.
 * The value 0 means that the location is not in mainland US
 */
static float nDeclination[] =
(
    0, 0, 4, 1, 0, 0, -8, 0, 0, 0, 0, 0, 0, 0, 0,
    0, 0, 0, 1, -3, -4, -6, -8, -10, -11, 12, -13, 0, 0,
    0, 0, 5, 2, -2, -5, -6, -9, -11, -12, -13, -13, -14,
    0, 7, 5, 2, -1, -4, -6, -8, -12, -12, -14, -15, -16, -16,
    0, 12, 9, 6, -1, -4, -6, -9, -11, -12, -14, -16, -17, -17,
    18, 15, 0, 7, 2, -3, -6, -9, -12, -13, -15, -18, -19, -19,
    0, 0, 0, 0, 0, -1, -6, -9, -13, -15, -18, -20, -21, -22
);

#endif LOG_ECB_ACTIVITY
int DPC_VOID;
LogHandle LogClientHandle;
EventHandle LogECBHandle;
#endif /* LOG_ECB_ACTIVITY */

/*
 * Local function prototypes.
 */
void ReturnResources(int sig);
.....
ExitHandler(void *handle)
{
    Description:
        This routine is called when the user hits Alt-F10 to exit
        the utility. We will attempt to verify with the user that
        they really want to exit.

    Input:
        handle - NUT handle

    Output:
        nothing

    Returns:
        nothing
    .....
}

void ExitHandler(void *handle)
{
    if (WMSConfirm(InxMSG("Exit DSDDEBUG?", 86), 0, 0, TRUE, NULL,
        handle, NULL) == TRUE)

```

```

{
    ReturnResources(1);
    exit(1);
}

static void NoSortHandler(LIST *head, LIST *tail, NUTInfo *handle)
{
    head = head;
    tail = tail;
    handle = handle;
    return;
}

.....
calcrc(WORD crc,
        BYTE *cp,
        LONG len)

Description:
    This routine calculates a CRC value for the text passed
    in. It uses a CRC substitution table for efficiency.
    Its used by the Slip transmit routine.

Input:
    crc - Initial value of the C
    cp - string to calc
    len - length of the string

Output:
    nothing

Returns:
    16-bit CRC value
    .....
WORD calcrc(WORD crc, BYTE *cp, LONG len)
{
    while(len--)
        crc = (crc >> 8) ^ circTab[(crc ^ *cp++) & 0xff];
    return(crc);
}

.....
SlipSend(char *pdata,
        LONG sz,
        int cas,
        int timeout)

Description:
    This routine builds a SLIP envelope around the message
    passed in, escapes any HDLC characters in the message,
    and sends it to the modem.

Input:
    pdata - Pointer to the message
    sz - length of the
    message passed in

```

```

* send to hub      cas      - 1 if send to cas, 0 if
*                 timeout   - inactivity timeout in seconds
*
* Output:
*         nothing
*
* Returns:
*         nothing
*
*.....
void SLIPSend( char *pdata, LONG sz, int cas, int timeout)
{
    LROBSPkt_t txcas;
    LROBSPkt_t txbub;
    LONG tmlen;
    WORD crc;
    BYTE *pi, *po;
    BYTE slip_data(3072);
    int is, os;

    if (cas)
    {
        /* Initialize cas transfer */
        CMovB(pdata, txcas.data, sz);
        txcas.length = sz + (txcas.data - ((BYTE *) &txcas));
        tmlen = txcas.length;
        txcas.length |= 0x8000;
        crc = calcrc(INITCRC, (BYTE *) &txcas, tmlen);
        txcas.data[sz] = crc & 0xff;
        txcas.data[sz+1] = (crc >> 8) & 0xff;
        pi = (BYTE *) &txcas;
    }
    else
    {
        /* Initialize hub transfer */
        CMovB(pdata, txbub.data, sz);
        txbub.length = sz + (txbub.data - ((BYTE *) &txbub));
        txbub.filler = txbub.channel = 0;
        txbub.control = 0x8000;
        tmlen = txbub.length;
        crc = calcrc(INITCRC, (BYTE *) &txbub, tmlen);
        txbub.data[sz] = crc & 0xff;
        txbub.data[sz+1] = (crc >> 8) & 0xff;
        pi = (BYTE *) &txbub;
    }

    /* Start out with SLIP start character */
    po = slip_data;
    *po++ = END;

    /* Escape any special SLIP characters that may be in the message */
    for (is = 0, os = 1; is < (tmlen + 2); is++, os++, pi++, po++)
    {
        switch(*pi)
        {
            case END:
                *po++ = ESC;
                *po = ESC_END;
                os++;
                break;
            case ESC:
                *po++ = ESC;
                *po = ESC_ESC;
                os++;
        }
    }
}

*.....
EXPORTED FUNCTION
DPCGetBoard(LONG *board,
             LONG *controlEntry)
{
    Description:
        This routine returns the DPC MLID logical board number
        and control entry address to be used to send IOCTL
        requests to the DPC mlid. The IOCTL mlid pragma can
        then be used to make the request.

    Input:
        board          - Pointer to where to at
        controlEntry   - Pointer to where to store control entr
    Y address

    Output:
        board and controlEntry filled in if successful

    Returns:
        0              if MLID is active
        *.....
LONG DPCGetBoard(LONG *board, LONG *controlEntry)
{
    if (DIOBoard == 0)
        return(-1);

    *board = DIOBoard;
    *controlEntry = DIOControlEntry;
    return(0);
}

void
CreateStringWithCommas( LONG number, BYTE *buffer, char *format )
{
    int i;
    int j;
    int found;
    int length;
    int commasAdded = 0;
    BYTE tmpBuf[ 128 ];
    BYTE *tmpPtr;

    nmsprintf( tmpBuf, format, number);
    for ( i = ( length = CStrLen( tmpBuf ) ), found = 0; i >= 0; i-- )

```

```

    {
        if ( ( tmpBuf[ i ] >= '0' ) && ( tmpBuf[ i ] <= '9' ) )
        {
            /* we have a digit */
            if ( ++found > 3 )
            {
                found = 1;

                /* shift the string one to the right */
                for ( j = ++length; j > i; j-- )
                    tmpBuf[ j ] = tmpBuf[ j - 1 ];

                tmpBuf[ i + 1 ] = GBIDOSCountryInfo.thousandSep[
                    0 ];

                commasAdded++;
            }

            /* Adjust the length of the string back to its original if there are
            ** leading spaces.
            */
            for ( i = 0, tmpPtr = tmpBuf; i < commasAdded; i++ )
            {
                if ( *tmpPtr == ' ' )
                    tmpPtr++;
            }

            strcpy( buffer, tmpPtr );
        }

        void
        SecondToDateAndTime( LONG seconds, BYTE *buff )
        {
            char *ascBuf;

            ascBuf = asctime(localtime((time_t *)&seconds));
            strncpy(ascBuf, buff, 26);
            buff[24] = 0;
        }

        void
        FormatElapsedTime( LONG seconds, LONG tenths, BYTE *buff )
        {
            LONG minutes, hours, days;

            /* convert secs to minutes */
            minutes = seconds / 60;
            seconds = seconds % 60;
            hours = minutes / 60;
            minutes = minutes % 60;
            days = hours / 24;
            hours = hours % 24;
            if ( days > 0 )
            {
                NMSprintf
                (
                    buff,
                    MSG("d %c%02d%c%02d%c%02d%c%01d", 293),
                    days,
                    GBIDOSCountryInfo.timeSep[ 0 ],
                    d,

```

```

h
    ** so we will use local copies of these variables adjusted to fit in wit
    ** the rest in the PCB structure.
    virtualHeight = portal->virtualHeight - 1;
    portalHeight = portal->portalHeight - 1;
    /*
    ** Other initializations.
    */
    bottomline = virtualHeight - portalHeight;

    while(!escapeFlag)
    {
        if ( updatedisplay == TRUE )
        {
            /*
            ** The last iteration of the loop made a change, so redr
            ** portal.
            */
            if ( portal->verticalScroll == SCROLL_ON )
            {
                /*
                ** Adjust the vertical thumb on the right border
                */
                if ( portal->cursorline == 0 )
                {
                    if ( virtualHeight <= portalHeight )
                        curPos = 100;
                    else
                        curPos = 0;
                }
                else
                {
                    if ( portal->cursorline >= bottomline )
                        curPos = 100;
                    else
                    {
                        vline = bottomline;
                        if ( vline == 0 )
                            vline = 1;
                        curPos = ( portal->cursorline *
100 ) / vline;
                    }
                }
            }
            /*
            ** Erase the vertical thumb at its last position
            */
            portal->showScrollBar &= -VERTICAL_SCROLL_MASK;
            /*
            ** Display the vertical thumb at its new positio
            */
            up. However, we need
        }
    }
    portal->showScrollBar |= curPos << VERTICAL_SCR
    NMSUpdatePortal( portal );
    updatedisplay = FALSE;
    switch ( keyType )
    {
        case K_UP:
            /*
            ** Scroll the portal up one line.
            */
            if ( portal->virtualline > 0 )
            {
                /*
                ** We're not at the top, so we can scrol
                */
                portal->virtualline--;
                portal->cursorline = portal->virtualline
                updatedisplay = TRUE;
            }
            else
            {
                /*
                ** Scroll the portal down one line.
                */
                if ( portal->virtualline < bottomline )
                {
                    /*
                    ** We're not at the bottom, so we can sc
                    */
                    portal->virtualline++;
                    portal->cursorline = portal->virtu
                    updatedisplay = TRUE;
                }
            }
            break;

            case K_PUP:
                /*
                ** Move the portal up one page.
                */
                if ( portal->cursorline > 0 )
                {
                    LONG delta;
                    /*
                    ** We're not at the top, so we can move
                }
            }
        }
    }

```



```

** to figure out how much we can move up
*/
if ( portal->cursorline > portalHeight )
    delta = portalHeight;
else
    delta = portal->cursorline;

portal->cursorline -= delta;
if ( portal->cursorline < portal->virtua
    portal->virtualline = portal->cu

    updatedDisplay = TRUE;

}

break;

case K_PDOWN:
    /*
    ** Move the portal down one page.
    */
    if ( portal->cursorline < virtualHeight )
    {
        LONG delta;
        LONG newCurrentline;

        /*
        ** We're not at the bottom, so we can mo
        ** we need to figure out how much we can
        */
        delta = virtualHeight - portal->cursorli
        if ( delta > portalHeight )
            delta = portalHeight;

        newCurrentline = portal->cursorline + de
        delta = virtualHeight - portalHeight;
        if ( newCurrentline > delta )
            portal->virtualline = delta;
        else
            portal->virtualline = newCurrent
        portal->cursorline = portal->virtualline
        updatedDisplay = TRUE;

    }
    break;

case K_SUP:
    /*
    ** <Ctrl-PgUp> takes us to the top of the portal
    */
    portal->virtualline = 0;
    portal->cursorline = 0;
    updatedDisplay = TRUE;

}

}

LONG DPCGetSignalStrength(void)
{
    struct DriverStatsStructure *stats;
    CUSTVARS *custompPtr;
    LONG signal;
    int beamSize, beamPercent;

    if (DPCGetMLIDStats(&stats))
        return(0);

    custompPtr = (CUSTVARS *)(&stats->CustomVariableCount);
    signal = custompPtr->CustomVariable(0);

    if (signal >= 200)
        signal = (2 * (signal - 200)) + 60;
    else
        signal = 0;

    beamSize = (int)((signal - 60L)/2L);
    beamSize *= 5;
    beamSize /= 4;

    if(beamSize < 0)
    {
        beamSize = 0;
    }
    else if(beamSize >= MAX_BEAM)
    {
        beamSize = MAX_BEAM;
    }

    beamPercent = (100*beamSize) / MAX_BEAM;
    return(beamPercent);

}

void UpdateStatsInformation(LONG portal)
{
    PCB *portalPcr;
    struct DriverStatsStructure *stats;

```

```

int line, count;
LONG numGenerics;
BYTE string[80];
LONG mask;
LONG seconds;
LONG tenths;
CustVars *customPtr;
BYTE *ptr;

NMSGGetPCB( aportalPtr, portal, NUTHandle );

if (DPCCGetMLIDStats(&stats) )
{
    AddKey( NUTHandle->screenID, ENTER_KEY, 0, 0, 0 );
    return;
}

/* do generic stuff */
line = GenericLineStart;
statsPtr = &( stats->NotSupportedMask );
numGenerics = stats->GenericVariableCount;
mask = *statsPtr++;

for ( count = 0; count < numGenerics; count++ )
{
    if ( mask & ( 0x80000000 >> ( count & 0x1f ) ) )
    {
        /* not supported */
        NMSGprintf( string, MSG("%13.13s", 301), MSG("Not support
ed", 298) );

        NMSGShowPortalLine
        (
            line++,
            GblDataCol,
            string,
            STATS_DATA_WIDTH,
            portalPtr
        );
        statsPtr++;
    }
    else
    {
        if ( count == 20 )
        {
            BYTE tmpString[ 80 ];

            /*
             * This is the operating time stamp, which needs
             * output format.
             */
            ConvertTicksToSeconds( *statsPtr++, &seconds, &t
enths );
            FormatElapsedTime( seconds, tenths, tmpString );
            NMSGprintf( string, MSG("%13.13s", 302), tmpStrin
g );
        }
        else
        {
            CreateStringWithCommas
            (
                *statsPtr++,
                string,
                MSG("%13lu", 303)
            );
            NMSGShowPortalLine
            (
                line++,
                GblDataCol,
                string,
                STATS_DATA_WIDTH,
                portalPtr
            );
        }
        /* do custom stuff */
        line += 2;
        customPtr = (CustVars *)(&stats->CustomVariableCount);
        ptr = (BYTE *)(&customPtr->CustomVariable[customPtr->CustomVariab
leCount]);
        ptr += sizeof(WORD);
        for ( count = 0; count < customPtr->CustomVariableCount; count++ )
        {
            CreateStringWithCommas
            (
                customPtr->CustomVariable[ count ],
                string,
                MSG("%13lu", 299)
            );
            NMSGShowPortalLine( line++, GblDataCol, string, STATS_DATA_WIDTH,
portalPtr );
        }
        NMSGUpdatePortal( portalPtr );
    }
}

void SignalMeter(void) {
    int exitNow = FALSE;
    LONG type;
    BYTE value;
    int signal = 0;
    int oldSignal = 0;
    int avgSqr = 0;
    int beamSize;
    int beamPercent;
    int block = FALSE;
    int rxFreq;
    struct DriverStatsStructure *stats;
    CustVars *customPtr;
    char freqStr[80];
    char avestr[80];
    char signalStr[80];
    int sound_gap = 0;

    while(!exitNow) {
        if (DPCCGetMLIDStats(&stats)) {
            type = signal = 0;
            rxFreq = 0;
        }
        else {
            customPtr = (CustVars *)(&stats->CustomVariableCount);
            type = signal = customPtr->CustomVariable[0];
        }
    }
}

```

```

    rxFreq = customPtr->CustomVariable[1];
}

NWsprintf(freqStr, MSG("    Frequency : %d", 345), rxFreq / 10);

if (signal >= 200)
    signal = (2 * (signal - 200)) + 60;
else
    signal = 0;

if (avgSqf == 0L)
    avgSqf = 100L * signal;
avgSqf = ((avgSqf * 19L) + (100L * (unsigned long) signal)) / 20L;

beamSize = (int)((avgSqf/100L - 60L)/2L);
beamSize *= 5;
beamSize /= 4;

if (beamSize < 0)
    beamSize = 0;
else if (beamSize >= MAX_BEAM)
    beamSize = MAX_BEAM - 1;

beamPercent = (100*beamSize) / MAX_BEAM;

if (oldSignal != signal) {
    if (signal < MIN_SQF_VAL)
        block = FALSE;
    else
        block = TRUE;
    oldSignal = signal;
}

NWsprintf(aveStr, MSG("    Average SQF : %d", 346),
    signal ? avgSqf / 100L : 0);

NWprintf(signalStr, MSG("Signal Quality : %d (%d%%)", &signal, 347),
    signal,
    beamPercent,
    block ? MSG("Signal Locked"), 348 : MSG("Signal Not Locked"), 349);

DisplayThrottle(MSG("Satellite Dish Signal Strength Meter", 341),
    beamSize,
    MAX_BEAM,
    freqStr,
    aveStr,
    signalStr);

if (--sound_gap <= 0) {
    /* calculate frequency based on raw signal strength */
    signal = 200 + type;
    /* adjust for 8253-5 timer chip output */
    signal = 1193180 / signal;

    /* turn on sound */
    outp(67, 182);
    outp(66, signal % 256);
    outp(66, signal / 256);
    outp(97, inp(97) | 0x03);

    delay(300);

    /* turn off sound */
    outp(97, inp(97) & ~0x03);
}
}

sound_gap = (110 - beamPercent) / 17;

delay(150);

if (NMWKeyStatus(NUTHandle)) {
    NMWGetKey(&type, &value, NUTHandle);
    if ((type == K_ESCAPE) || (type == K_AF10))
        exitNow = TRUE;
}

/* Destroy the throttle portal */
DisplayThrottle(MSG("Satellite Dish Signal Strength Meter", 351),
    MAX_BEAM,
    MAX_BEAM,
    freqStr,
    aveStr,
    signalStr);
}

void GetRegionName(char *regionName)
{
    FILE *fp;
    char szTmp[128];
    char *src, *dest;
    LONG len;

    fp = fopen(MSG("country.ini", 635), MSG("r", 636));
    len = strlen(MSG("RegionName=", 637));
    if (fp != NULL)
    {
        while (fgetc(szTmp, sizeof(szTmp) - 1, fp) != NULL)
        {
            if ( (strlen(cmp(szTmp, MSG("RegionName=", 638), len)) ==
                fclose(fp);
                src = &szTmp[len];
                dest = regionName;
                while(*src != 0 && *src != ' ' && *src != '\r' &
                    {
                        *dest = *src;
                        src++;
                        dest++;
                    }
                *dest = 0;
                return;
            }
            fclose(fp);
        }
    }

    void GetCountry(char *countryName)
    {
        FILE *fp;
        char szTmp[128];
        char *src, *dest;
        LONG len;
    }
}

```

```

fp = fopen(MSG("country.in", 639), MSG("r", 640));
len = strlen(MSG("Name=", 641));
*countryName = 0;
if (fp != NULL)
{
    while (fgets(szTmp, sizeof(szTmp) - 1, fp) != NULL)
    {
        if ( (strcmp(szTmp, MSG("Name=", 642), len) == 0)
        {
            fclose(fp);
            src = szTmp[len];
            dest = countryName;
            while(*src != 0 && *src != ' ' && *src != '\r' &
            {
                *dest = *src;
                src++;
                dest++;
            }
            *dest = 0;
            return;
        }
    }
    fclose(fp);
}

typedef struct
{
    FILE *fp;
    char szTmp[128];
    char *src, *dest;
    LONG len;

    fp = fopen(MSG("country.in", 626), MSG("r", 627));
    len = strlen(MSG("Service=", 628));
    if (fp != NULL)
    {
        while (fgets(szTmp, sizeof(szTmp) - 1, fp) != NULL)
        {
            if ( (strcmp(szTmp, MSG("Service=", 185), len) == 0)
            {
                fclose(fp);
                src = szTmp[len];
                dest = serviceName;
                while(*src != 0 && *src != ' ' && *src != '\r' &
                {
                    *dest = *src;
                    src++;
                    dest++;
                }
                *dest = 0;
                return;
            }
        }
        fclose(fp);
    }
}

typedef struct
{
    char name[20];
    LONG longitude;
    LONG eastFlag;
    LONG frequency;
    LONG horzFlag;
    LONG cityLongDegrees;
    LONG cityLongMinutes;
    LONG cityLatMinutes;
    LONG cityEastFlag;
    LONG cityNorthFlag;

    SatelliteInfo;

    typedef struct
    {
        float elevation;
        float trueAzimuth;
        float magAzimuth;
        float polarization;
    } DishInfo;

    void ParseSatelliteInfo(char *satName, int nameContainsInfo, SatelliteInfo *sat)
    {
        FILE *fp;
        char szTmp[128];
        char *fptr;
        char *name, *cptr;
        char *ascPtr;
        char ascBuf[10];

        if (nameContainsInfo == FALSE)
        {
            fp = fopen(MSG("country.in", 644), MSG("r", 629));
            if (fp != NULL)
            {
                while (fptr = fgets(szTmp, sizeof(szTmp) - 1, fp) != N
                {
                    if ( (strcmp(szTmp, satName, strlen(satName)))
                    {
                        break;
                    }
                    if (fp)
                    {
                        fclose(fp);
                        if (!fptr)
                        {
                            return;
                        }
                        cptr = szTmp;
                    }
                    else
                    {
                        cptr = satName;
                    }
                    name = sat->name;
                    while(*cptr != '=')
                    {
                        *name++ = *cptr++;
                    }
                    *name = 0;
                    cptr++;
                    while(*cptr == ' ')
                    {
                        cptr++;
                    }
                    ascPtr = ascBuf;
                    while(*cptr != ',')
                    {
                        *ascPtr++ = *cptr++;
                    }

```

```

cptr++;
*ascPtr = 0;
sat->longitude = atoi(ascBuf);

if (*cptr == 'e' || *cptr == 'E')
    sat->eastFlag = 1;
else
    sat->eastFlag = 0;

while(*cptr != ',')
    cptr++;

ascPtr = ascBuf;
while(*cptr != ',')
    *ascPtr++ = *cptr++;
cptr++;
*ascPtr = 0;
sat->frequency = atoi(ascBuf);

if (*cptr == 'h' || *cptr == 'H')
    sat->horzFlag = 1;
else
    sat->horzFlag = 0;

)

void GetDefaultSatellite(SatelliteInfo *sat)
{
    FILE *fp;
    char szTmp[128];
    LONG len;
    char *ccode;

    fp = fopen(MSG("country.ini", 632), MSG("r", 633));
    len = strlen(MSG("Hughes Networks", 630));
    if (fp != NULL)
    {
        while (fgets(szTmp, sizeof(szTmp) - 1, fp) != NULL)
        {
            if ( (strlen(szTmp, MSG("Hughes Networks", 631), len)
                == 0)
            {
                ccode = fgets(szTmp, sizeof(szTmp) - 1, fp);
                fclose(fp);
                if (ccode == NULL)
                    return;

                ParseSatelliteInfo(szTmp, TRUE, sat);
                return;
            }
        }
        fclose(fp);
    }

    int NewCalculationFlag = 0;

    LONG ChangesSatellite(FIELD *fieldPtr, int key, int *changed, NUTInfo *handle)
    {
        FILE *fp;
        SatelliteInfo *sat;
        LIST *listPtr = NULL;
        LONG ccode;
        LONG rcode = K_SELECT;

        char *fstr;
        char szTmp[128];
        char *szPtr;
        int rows = 0, cols = 0, len;
        void (*oldSortFunction)(LIST *, LIST *, NUTInfo *);

        key = key;
        changed = changed;
        handle = handle;

        sat = (SatelliteInfo *)fieldPtr->customData;

        MMSGetListSortFunction(NUTHandle, kOldSortFunction);
        MMSSetListSortFunction(NUTHandle, NoSortHandler);

        if (MMSPushList(NUTHandle) == 0)
        {
            MMSSetListSortFunction(NUTHandle, oldSortFunction);
            return rcode;
        }

        MMSInitList(NUTHandle, NULL);

        fp = fopen(MSG("country.ini", 634), MSG("r", 667));
        len = strlen(MSG("Hughes Networks", 668));
        if (fp != NULL)
        {
            while ((fstr = fgets(szTmp, sizeof(szTmp) - 1, fp)) != NULL)
            {
                if ( (strlen(szTmp, MSG("Hughes Networks", 205), len)
                    == 0)
                )
                {
                    break;
                }
                if (fstr != NULL)
                {
                    while (fgets(szTmp, sizeof(szTmp) - 1, fp) != NULL)
                    {
                        if (*szTmp == ' ' || *szTmp == '\r' || *szTmp == '\n')
                            break;
                        szPtr = szTmp;
                        while (*szPtr != '=')
                            szPtr++;
                        *szPtr = 0;
                        AppendToList(szTmp, 0, &rows, &cols);
                    }
                }
                if (rows == 0)
                    goto ChangesSatellite;

                ccode = MMSList(
                    InMSG("Choose Satellite", 648),
                    12, 40,
                    (rows > 16 ? 16 : rows,
                     (cols < 20 ? 20 : cols,
                      M_ESCAPE | M_SELECT,
                      &listPtr,
                      NUTHandle, NULL,
                      NULL, NULL);

                if (ccode == M_SELECT)
                {
                    ParseSatelliteInfo(listPtr->text, FALSE, sat);

```



```

(
    LD = floor(LD/4) - 6;
    RD = ceil(RD/4) - 18;
    if(!inDeclination((int)(LD*14+RD)))
        count--;
    if(!inDeclination((int)(LD*14+RD-1)))
        count--;
    if(!inDeclination((int)((LD+1)*14+RD-1)))
        count--;
    if(!inDeclination((int)((LD+1)*14+RD)))
        count--;
    if(count)
    {
        AZ = AZ + inDeclination((int)(LD*14+RD))
        + inDeclination((int)(LD*14+RD-1))
        + inDeclination((int)((LD+1)*14+RD))
        + inDeclination((int)((LD+1)*14+RD-1))/count;
    }
}

if (AZ >= 360) AZ = AZ - 360;
if (AZ < 0) AZ = AZ + 360;

S = tan(LN) / sin(LW);

PO = atan(S);
/* printf("S=%f if LN=%f if LW=%f if PO=%f\n", S, LN, LW, PO); */

PO = fabs(PO);
PO = M_PI / 2.0 - PO;
PO = PO * SGN(S) * 180 / M_PI;
PO = floor(10 * PO + .5 * SGN(PO)) / 10;

p->dremMagAz = EL;
p->dremMagAz = AZ;
p->dremTrueAz = TRAZ;
p->dremPolar = -PO;
dish->elevation = EL;
dish->magAzimuth = AZ;
dish->trueAzimuth = TRAZ;
dish->polarization = -PO;

}

void ComputeCity(char *region, char *city, LONG latLong)
{
    FIELD *fp;
    char country[20], countryStr[30];
    char regionStr[30], cityStr[30];
    char service[30], serviceStr[30];
    char satelliteStr[50];
    char elevationStr[50];
    char azimuthStr[50], trueAzimuthStr[50];
    char magAzimuthStr[50], polarizationStr[50];
    satelliteInfo sat;
    DishInfo dish;
    int i;
    int start;
    MSGCONTROL *mfc10, *mfc11, *mfc12, *mfc13;

    calculateAgain:
    NWSInitForm(NUTHandle);
    if (NewCalculationFlag == 0)
{
    sat.cityLatDegrees = (latLong >> 24) & 0xff;
    sat.cityLatMinutes = (latLong >> 16) & 0xff;
    sat.cityLongDegrees = (latLong >> 8) & 0xff;
    sat.cityLongMinutes = latLong & 0xff;
    GetDefaultSatellite(&sat);
    sat.cityEastFlag = sat.eastFlag;
    sat.cityNorthFlag = 1;
}
    NewCalculationFlag = 0;

    i = 0;
    GetCountry(country);
    NWSprintf(countryStr, MSG("Country : %s", 488), country);
    NWSAppendCommentField(i, 2, countryStr, NUTHandle);

    NWSprintf(regionStr, MSG("Region : %s", 712), region);
    NWSAppendCommentField(i, 36, regionStr, NUTHandle);

    i++;
    NWSprintf(cityStr, MSG("City : %s", 713), city);
    NWSAppendCommentField(i, 2, cityStr, NUTHandle);

    GetDefaultService(service);
    NWSprintf(serviceStr, MSG("Service : %s", 714), service);
    NWSAppendCommentField(i, 36, serviceStr, NUTHandle);

    i++;
    NWSprintf(satelliteStr, MSG("Satellite : %s", 649), sat.name);
    start = (78 - strlen(satelliteStr)) / 2;
    fp = NWSAppendHotspotField(i, start, NORMAL_FIELD, satelliteStr,
        ChangesSatellite, NUTHandle);
    fp->customData = &sat;

    i++;
    NWSAppendCommentField(i, 2, MSG("Satellite Longitude : ", 715), NUTHandle);
    NWSAppendIntegerField(i, 27, NORMAL_FIELD, (int *)&sat.longitude, 1, 180,
        F_NO_HELP, NUTHandle);

    NWSAppendCommentField(i, 34, MSG("Hemisphere : ", 716), NUTHandle);
    mfc10 = NWSInitMenuField(inxMSG("Satellite Longitude Hemisphere", 717),
        10, 40, LongitudeHemisphereHandler, NUTHandle);
    NWSAppendMenuItemField(mfc10, inxMSG("West", 718), 0, NUTHandle);
    NWSAppendMenuItemField(mfc10, inxMSG("East", 719), 1, NUTHandle);
    NWSAppendMenuItemField(i, 47, NORMAL_FIELD, (int *)&sat.eastFlag, mfc10, NUTHandle);

    i++;
    NWSAppendCommentField(i, 2, MSG("Satellite Polarization : ", 489), NUTHandle);
    mfc11 = NWSInitMenuField(inxMSG("Satellite Polarization", 490), 10, 40,
        PolarizationHandler, NUTHandle);
    NWSAppendMenuItemField(mfc11, inxMSG("Vert", 531), 0, NUTHandle);
    NWSAppendMenuItemField(mfc11, inxMSG("Horz", 539), 1, NUTHandle);
    NWSAppendMenuItemField(i, 27, NORMAL_FIELD, (int *)&sat.horzFlag, mfc11, NUTHandle);

    NWSAppendCommentField(i, 34, MSG("Frequency : ", 541), NUTHandle);
    NWSAppendIntegerField(i, 47, NORMAL_FIELD, (int *)&sat.frequency, 1, 100,
        F_NO_HELP, NUTHandle);

    i++;
    NWSAppendCommentField(i, 2, MSG("Ground Longitude degrees : ", 543), NUTHandle);
    NWSAppendIntegerField(i, 30, NORMAL_FIELD, (int *)&sat.cityLongDegrees,
        1, 180, F_NO_HELP, NUTHandle);
}
}

```

```

NMSAppendCommentField(i, 40, MSG("minutes : ", 545), NUTHandle);
NMSAppendIntegerField(i, 50, NORMAL_FIELD, (int *)ksat.cityLongMinutes,
F_NO_HELP, NUTHandle);
1, 180, F_NO_HELP, NUTHandle);
NMSAppendCommentField(i, 58, MSG("Hemisphere : ", 686), NUTHandle);
mfc12 = NMSInitMenuField(InxMSG("Ground Longitude Hemisphere", 687), 10
, 40, GroundLongHemHandler, NUTHandle);
NMSAppendMenuField(mfc12, InxMSG("West", 688), 0, NUTHandle);
NMSAppendMenuField(mfc12, InxMSG("East", 689), 1, NUTHandle);
NMSAppendMenuField(i, 71, NORMAL_FIELD, (int *)ksat.cityEastFlag, mfc12
, NULL, NUTHandle);
1++;
NMSAppendCommentField(i, 2, MSG("Ground Latitude degrees : ", 690), NUT
Handle);
NMSAppendIntegerField(i, 30, NORMAL_FIELD, (int *)ksat.cityLatDegrees, 1
, 180, F_NO_HELP, NUTHandle);
NMSAppendCommentField(i, 40, MSG("minutes : ", 691), NUTHandle);
NMSAppendIntegerField(i, 50, NORMAL_FIELD, (int *)ksat.cityLatMinutes, 1
, 180, F_NO_HELP, NUTHandle);
NMSAppendCommentField(i, 58, MSG("Hemisphere : ", 692), NUTHandle);
mfc13 = NMSInitMenuField(InxMSG("Ground Latitude Hemisphere", 693), 10
, 40, GroundLatHemHandler, NUTHandle);
NMSAppendMenuField(mfc13, InxMSG("South", 694), 0, NUTHandle);
NMSAppendMenuField(mfc13, InxMSG("North", 695), 1, NUTHandle);
NMSAppendMenuField(i, 71, NORMAL_FIELD, (int *)ksat.cityNorthFlag, mfc1
3, NULL, NUTHandle);
1++;
NMSAppendHotSpotField(i, 35, NORMAL_FIELD, MSG("COMPUTE NOW", 696),
ComputeHotSpot, NUTHandle);
CalculatedDish(ksat, &dish);
{
if (sat.horzflag == 0)
{
dish.polarization += 90.0;
if (dish.polarization > 90.0)
dish.polarization -= 90.0;
if (dish.polarization < -90.0)
dish.polarization += 90.0;
}
1+=2;
NMSprintf(elevationStr, MSG(" Elevation : %lf", 697), dish.elev
ation);
NMSprintf(trueAzimuthStr, MSG(" True Azimuth : %lf", 698), dish.true
Azimuth);
if ((strcmp(country, MSG("USA", 243))) != 0)
strcpy(magAzimuthStr, MSG("Magnetic Azimuth : N/A", 699));
else
NMSprintf(magAzimuthStr, MSG("Magnetic Azimuth : %lf", 568), d
ish.magAzimuth);
NMSprintf(polarizationStr, MSG(" Polarization : %lf", 700), dish.pola
rization);
NMSAppendCommentField(i, 2, elevationStr, NUTHandle);
1++;
NMSAppendCommentField(i, 2, trueAzimuthStr, NUTHandle);
1++;
NMSAppendCommentField(i, 2, magAzimuthStr, NUTHandle);
1++;
}

void ComputeGeCity(char *region)
{
DIR *dirCountry, *dirCity;
char *name;
LIST *listPtr = NULL;
LONG ccode = M_SELECT;
int rows = 0, cols = 0;
FILE *fp;
char szTmp[128];
char bytesStr[8];
int latDeg, latMin, longDeg, longMin;
int len, start = 0;
LONG info;

NMSStartWait(0, 0, NUTHandle); /* DMH change 970131 */
getCityLoop:
rows = cols = 0;
listPtr = NULL;
NMSInitList(NUTHandle, NULL);
dirCountry = opendir(MSG("*.cty", 702));
if (dirCountry == NULL)
goto errorNodir;
while ( (dirCity = readdir(dirCountry)) != NULL )
{
name = dirCity->d_name;
fp = fopen(name, MSG("r", 703));
if (fp == NULL)
{
closedir(dirCountry);
goto errorNodir;
}
len = strlen(region);
if (fgets(szTmp, sizeof(szTmp) - 1, fp) != NULL)
if ( (strcmp(region, szTmp, len) == 0)
fclose(fp);
}
}

```



```

    if (dirCltty == NULL)
    {
        closedir(dirCountry);
        goto errorNodir;
    }

    while (fgets(szTmp, sizeof(szTmp) - 1, fp) != NULL)
    {
        int len = strlen(szTmp) - 2;
        byteStr[2] = szTmp[len--];
        byteStr[1] = szTmp[len--];
        byteStr[3] = 0;
        longMin = atoi(&byteStr[1]);

        byteStr[2] = szTmp[len--];
        byteStr[1] = szTmp[len--];
        byteStr[0] = szTmp[len--];
        longDeg = atoi(&byteStr);

        byteStr[2] = szTmp[len--];
        byteStr[1] = szTmp[len--];
        latMin = atoi(&byteStr[1]);

        byteStr[2] = szTmp[len--];
        byteStr[1] = szTmp[len--];
        latDeg = atoi(&byteStr[1]);

        if (longMin > 59)
        {
            longDeg++;
            longMin = 0;
        }

        if (latMin > 59)
        {
            latDeg++;
            latMin = 0;
        }

        info = ((latDeg & 0xff) << 24) |
                ((latMin & 0xff) << 16) |
                ((longDeg & 0xff) << 8) |
                (longMin & 0xff);

        while (szTmp[len] == ' ')
            len--;
        {
            szTmp[len+1] = 0;

            while (len)
            {
                if (szTmp[len] == ' ')
                    start = len + 1;
                len--;
            }
            if (start > 0)
                AppendToList(&szTmp[start], info, &rows, &cols);
        }
    }
}

fclose(fp);
if (rows == 0)
{
    errorNodir:
        NMSendWait( NUTHandle );
        ccode = NMSList(
            InxMSG("Choose A CItty", 704),
            12, 40,
            (rows < 16) ? rows : 16,
            (cols < 18) ? 18 : cols,
            M_ESCAPE | M_SELECT,
            listPtr,
            NUTHandle, NULL,
            NULL, NULL);

        if (ccode == M_SELECT)
        {
            info = (LONG)listPtr->otherInfo;
            strcpy(szTmp, listPtr->text);
            NMSdestroyList(NUTHandle);
            closedir(dirCountry);
            ComputeCltty(region, szTmp, info);
            goto getClttyLoop;
        }

        NMSdestroyList(NUTHandle);
        closedir(dirCountry);
        return;

    errorNodir:
        NMSendWait( NUTHandle );
        NMSdestroyList(NUTHandle);
        NMSAlert(12, 40, NUTHandle, InxMSG("Unable to access CItties file", 705))
        return;
    }

    void ComputeGetRegion(char *country)
    {
        DIR *dirCountry, *dirCltty;
        char *name, *end;
        LIST *listPtr = NULL;
        LONG ccode = M_SELECT;
        int rows = 0, cols = 0;
        char path[64];
        FILE *fp;
        char szTmp[128];
        LONG header;

        NMSStartWait( 0, 0, NUTHandle );

        getRegionsLoop:
            rows = cols = 0;
            listPtr = NULL;
            NMSinitList(NUTHandle, NULL);
            SetCurrentNameSpace(DOSNameSpace);
            NMSprintf(path, MSG("SYS:DIRPCP\\DB\\%s.cou", 706), country);
            if (chdir(path))
                goto errorNodir;

            dirCountry = opendir(MSG("*.cty", 707));
            if (dirCountry == NULL)
                goto errorNodir;

```

```

while( (dirCty = readdir(dirCountry)) != NULL )
{
    name = dirCty->d_name;

    fp = fopen(name, MSG("r", 708));
    if (fp == NULL)
    {
        closedir(dirCountry);
        goto errorNodir;
    }

    if(fgets(szTmp, sizeof(szTmp) - 1, fp) != NULL)
    {
        end = &szTmp[strlen(szTmp) - 2];
        while( ('end' == ' ') && (end != szTmp) )
            end--;
        end++;
        *end = 0;
        AppendToList(szTmp, 0, &rows, &cols);
        fclose(fp);
    }

    if (rows == 0)
    {
        closedir(dirCountry);
        goto errorNodir;
    }

    GetRegionName(szTmp);
    if ( (strcmp(szTmp, MSG("Province", 709))) == 0 )
        header = InxMSG("Choose A Province", 710);
    else
        header = InxMSG("Choose A State", 711);

    if (rows == 1)
    {
        NMSendWait( NUTHandle ); /* DMH change 970131 */
        NMSDestroyList(NUTHandle);
        ComputeGetCty(szTmp);
        closedir(dirCountry);
        return;
    }
    else
    {
        NMSendWait( NUTHandle ); /* DMH change 970131 */
        ccode = NMSList(
            header,
            12, 40,
            (rows < 16) ? rows : 16,
            (cols < 18) ? 18 : cols,
            M_ESCAPE | M_SELECT,
            &listPtr,
            NUTHandle, NULL,
            NULL, NULL);

        if (ccode == M_SELECT)
        {
            strcpy(szTmp, listPtr->text);
            NMSDestroyList(NUTHandle);
            closedir(dirCountry);
            ComputeGetCty(szTmp);
            goto getRegionsLoop;
        }
    }

    NMSDestroyList(NUTHandle);
    closedir(dirCountry);
    return;

errorNodir:
    NMSendWait( NUTHandle ); /* DMH change 970131 */
    NMSDestroyList(NUTHandle);
    NMSAlert(12, 40, NUTHandle, InxMSG("Unable to locate Country files in Co
untry directory %s.cou", 569), country);
    return;
}

void ComputeCoordinates(void)
{
    DIR *dirDB, *dirCountry;
    char *name, *dot;
    LIST *listPtr = NULL;
    LONG ccode = M_SELECT;
    int rows = 0, cols = 0;

    getCountriesLoop:
    rows = cols = 0;
    listPtr = NULL;
    NMSInitList(NUTHandle, NULL);
    SetCurrentNameSpace(DOSNameSpace);
    if (chdir(MSG("SYS:DIRECTPC\\DB", 570)))
        goto errorNodir;

    dirDB = opendir(MSG("**.cou", 571));
    if (dirDB == NULL)
        goto errorNodir;

    while( (dirCountry = readdir(dirDB)) != NULL )
    {
        name = dirCountry->d_name;
        if ( (dot = strchr(name, '.')) != NULL )
            *dot = 0;
        /* we don't want the extension */
        AppendToList(name, 0, &rows, &cols);
    }

    if (rows == 0)
    {
        closedir(dirDB);
        goto errorNodir;
    }

    ccode = NMSList(
        InxMSG("Choose A Country", 572),
        12, 40,
        rows,
        18,
        M_ESCAPE | M_SELECT,
        &listPtr,
        NUTHandle, NULL,
        NULL, NULL);

    if (ccode == M_SELECT)
    {
        if (NMSPushList(NUTHandle) != 0)
        {
            name = listPtr->text;

```

```

/* Height */
/* Width */

```



```

    for (col = 8; col <= 64; col+=16, count--)
    {
        if (!count)
            break;

        MMSprintf( string, MSG("%2.2x%2.2x%2.2x", 504),
            dptr->groupid.i[2], dptr->groupid.i[1],
            dptr->groupid.i[0], dptr->version);

        MMShowPortalLine(
            line,
            col,
            string,
            CStrlen(string),
            portalPtr
        );
        dptr++;
    }

    MMSUpdatePortal( portalPtr );
}

void DisplayAdapterInfo(void)
{
    int         line, len;
    BYTE        oldPortal;
    PCB         *portalPtr;
    BYTE        string[80];

    line = 5 + 3; /* Site ID, S/N, Key Status, Number of Communities,
                  Current Communities */
    extra community lines */
    line += (CASDbdcau.entries / 4);

    oldPortal = NUTHandle->currentPortal;
    MMSDeselectPortal( NUTHandle );
    BackgroundPortal = MMSCreatePortal
    (
        1 /* gblUPFLeftCol *//,
        1 /* gblUPFLeftCol *//,
        ((line + 4) > (ScreenHeight - 3)) ? ScreenHeight - 3 : line + 4,
        /* gblUPFHeight + gblUPFHeight *//,
        ScreenWidth - 2 /* gblUPFWidth *//,
        line,
        ( ScreenWidth - 4 /* gblUPFWidth - 2 */ ),
        TRUE,
        MSG("DPC Adapter Information", 502),
        VNORMAL,
        SINGLE,
        VINTENSE,
        CURSOR_OFF,
        VIRTUAL,
        NUTHandle
    );
    if ( BackgroundPortal > MAXPORTALS )
    {
        MMSselectPortal( oldPortal, NUTHandle );
        return;
    }
    MMSGetPCB( &portalPtr, BackgroundPortal, NUTHandle );
    portalPtr->showScrollBar = SHOW_VERTICAL_SCROLL_BAR;
    portalPtr->showScrollBar |= TEXT_SENSITIVE_SCROLL_BARS;
    portalPtr->verticalScroll = SCROLL_ON;
}

/*
 * Start filling in the static portal lines.
 */
line = 0;
MMSprintf(string, MSG("Serial Number : ", 496));
len = CStrlen( string );
MMSshowPortalLine( line++, BORDER_WIDTH, string, len, portalPtr );
MMSprintf(string, MSG("Site ID : ", 497));
len = CStrlen( string );
MMSshowPortalLine( line++, BORDER_WIDTH, string, len, portalPtr );
MMSprintf(string, MSG("Have Keys Status : ", 498));
len = CStrlen( string );
MMSshowPortalLine( line++, BORDER_WIDTH, string, len, portalPtr );
MMSprintf(string, MSG("Number Of Communities : ", 499));
len = CStrlen( string );
MMSshowPortalLine( line++, BORDER_WIDTH, string, len, portalPtr );
MMSprintf(string, MSG("Communities : ", 500));
len = CStrlen( string );
MMSshowPortalLine( line++, BORDER_WIDTH, string, len, portalPtr );

GblDataCol = BORDER_WIDTH + 24;

UpdateAdapterInformation(BackgroundPortal);
BackgroundFuncPtr = UpdateAdapterInformation;
HandleScrollablePortal(portalPtr);
BackgroundFuncPtr = NULL;
MMSDestroyPortal( BackgroundPortal, NUTHandle );
MMSselectPortal( oldPortal, NUTHandle );
}

void DisplayMIDStats(void)
{
    int         line;
    int         count;
    int         numberOfGenerics;
    int         len;
    int         promptMax;
    struct DriverStatsStructure *stats;
    struct DriverConfigurationStructure *config;
    ProtocolNodeStructure *protocol;
    CusctVars   *custompPtr;
    BYTE        *customStrings, *ptr;
    BYTE        oldPortal;
    BYTE        name[128], *namePtr;
    BYTE        PCB
    BYTE        string[80];

    GblDataCol = ( ScreenWidth - 4 ) - BORDER_WIDTH - STATS_DATA_WIDTH;
    if (DPCGetMIDStats(&stats))
    {
        return;
    }
    DIOGetMIDconfig(&config);

    /*
     * Build up the LAN name followed by its I/O resources

```

*/ to be used as the portals header.
*/

```
if (config->DLogicalName[0] != 0)
    NWSprintf(name, MSG("s %s", 270), config->DLogicalName, config
->DShortName);
else
```

```
    NWSprintf(name, MSG("s", 271), config->DShortName);
```

```
if (config->DIOPortsAndRanges[1] > 0)
```

```
{
    namePtr = name + CStrlen(name);
    NWSprintf(namePtr, MSG(" port=%X", 272), config->DIOPortsAndRang
```

```
es[0]);
```

```
if (config->DIOPortsAndRanges[3] > 0)
```

```
{
    namePtr = name + CStrlen(name);
    NWSprintf(namePtr, MSG(" port=%X", 273), config->DIOPortsAndRang
```

```
es[2]);
```

```
if (config->DMemoryDecodeAndLength[0].LANMemoryAddress > 0 )
```

```
{
    namePtr = name + CStrlen(name);
    NWSprintf(namePtr, MSG(" memory=%X", 274),
```

```
    config->DMemoryDecodeAndLength[0].LANMemoryAddre
```

```
ss);
```

```
if (config->DMemoryDecodeAndLength[1].LANMemoryAddress > 0 )
```

```
{
    namePtr = name + CStrlen(name);
    NWSprintf(namePtr, MSG(" memory=%X", 275),
```

```
    config->DMemoryDecodeAndLength[1].LANMemoryAddre
```

```
ss);
```

```
if (config->DIncline[0] != 0xff)
```

```
{
    namePtr = name + CStrlen(name);
    NWSprintf(namePtr, MSG(" int=%X", 276), config->DIncline[0]);
```

```
if (config->DIncline[1] != 0xff)
```

```
{
    namePtr = name + CStrlen(name);
    NWSprintf(namePtr, MSG(" int=%X", 277), config->DIncline[1]);
```

```
if (config->DDMAline[0] != 0xff)
```

```
{
    namePtr = name + CStrlen(name);
    NWSprintf(namePtr, MSG(" dma=%X", 278), config->DDMAline[0]);
```

```
if (config->DDMAline[1] != 0xff)
```

```
{
    namePtr = name + CStrlen(name);
    NWSprintf(namePtr, MSG(" dma=%X", 279), config->DDMAline[1]);
```

```
if (config->DMediaType != NULL)
```

```
{
    namePtr = name + CStrlen(name);
    NWSprintf(namePtr, MSG(" frame=%S", 280), config->DMediaType);
```

```
namePtr = name + CStrlen(name);
```

```
NWSprintf(namePtr, MSG(" ", 281));
```

/*
* Before we can create the portal, we must add up the number number
* of lines we'll need, which will be bigger than the window.

*/

line = 3;
protocol header */
/* lines for version, node address, and

/* Add up the number of protocols bound to this board */

```
protocol = MLIProtocolListByBoard( DIOBoard );
while (protocol != NULL)
```

```
{
```

```
    ++line;
```

```
    protocol = (ProtocolNodeStructure *)protocol->ProtocolBoardLink;
```

/* Add in the number of generic statistics */

```
line += 2; /* Blank line and Generic statistics header */
```

```
numberOfGenerics = stats->GenericVariableCount;
```

```
line += numberOfGenerics;
```

```
promptMax = 0;
```

```
for (count = 0; count < numberOfGenerics ; count++)
```

```
{
    len = CStrlen( GetMsg(GenericDescriptionTable[count]) );
    if (promptMax < len)
```

```
{
    promptMax = len;
```

```
}
```

/* Add in the number of custom statistics */

```
line += 2; /* Blank line and Custom statistics header */
```

```
customPtr = (CustVars *) (stats->CustomVariableCount);
```

```
customStrings = (BYTE *) (customPtr->CustomVariable(customPtr->CustomVar
```

```
iableCount));
customStrings += sizeof(WORD);
```

```
line += customPtr->CustomVariableCount;
```

/* Check lengths of custom strings */

```
ptr = customStrings; /* temp pointer to walk down custom prom
```

```
for ( count = 0; count < customPtr->CustomVariableCount; count++
```

```
{
    len = CStrlen( ptr );
    if ( promptMax < len )
```

```
    promptMax = len;
    while ( *(ptr++) )
```

```
        ; /* find the next string */
```

```
line += 1; /* add a blank line for the bottom */
```

```
if ( promptMax < 46 ) /* minimum portal width */
```

```
else if ( promptMax > 60 ) /* maximum portal width */
```

```
else
    promptMax += 16; /* custom portal width within range */
```

```
/* build the portal */
```

```

if ( line < 8 )
    line = 8;
/* must meet the minimum portal size */

oldPortal = NUTHandle->currentPortal;
NMWSelectPortal( NUTHandle );
BackgroundPortal = NMWCreatePortal
(
    1
    /* gblUPFtoPline */
    1 /* gblUPFtoCol */
    ScreenHeight - 3 /* gblUPFheight + gblUPFheight */
    ScreenWidth - 2 /* gblUPFwidth */
    line
    ( ScreenWidth - 4 /* gblUPFwidth - 2 */
    TRUE
    name
    VNORMAL
    SINGLE
    VINTENSE
    CURSOR_OFF
    VIRTUAL
    NUTHandle
);
if ( BackgroundPortal > MAXPORTALS )
{
    NMWSelectPortal( oldPortal, NUTHandle );
    return;
}
NMWGetPCB( &portalPtr, BackgroundPortal, NUTHandle );
portalPtr->showScrollBar = SHOW_VERTICAL_SCROLL_BAR;
portalPtr->showScrollBar |= TEXT_SENSITIVE_SCROLL_BARS;
portalPtr->verticalScroll = SCROLL_ON;
NMWClearPortal( portalPtr );

/*
 * Start filling in the static portal lines.
 */

/* Fill in the MLID version */
line = 0;
NMWprintf( string, MSG( "Version %d.%d", 282 ), config->DMLID_MajorVersion,
config->DMLID_MinorVersion );
len = CStrlen( string );
NMWShowPortalLine( line++, BORDER_WIDTH, string, len, portalPtr );

/* Fill in the node address */
NMWprintf( string, MSG( "Node Address: %X.%X", 283 ),
GET_HILO_LONG( (LONG *) &config->DNodeAddress[0] ),
GET_HILO_WORD( (WORD *) &config->DNodeAddress[4] ) );
len = CStrlen( string );
NMWShowPortalLine( line++, BORDER_WIDTH, string, len, portalPtr );

/* protocols */
NMWShowPortalLine( line++, BORDER_WIDTH, MSG( "Protocols:", 284 ),
CStrlen( MSG( "Protocols:", 287 ) ), portalPtr );
protocol = MLIDProtocolListByBoard( DIOBoard );
while ( protocol != NULL )
{
    GetProtocolNameTableEntry( protocol->ProtocolNumber, string );
    CMOV( ProtocolNameTable[protocol->ProtocolNumber], string, 16 );
    len = string[ 0 ];
    string[ len + 1 ] = NULL;
    NMWShowPortalLine

```



```

    , 551),

    SendModemRoutine, NUTHandle);

    i++;

    NMSeditPortalForm(InxMSG("Modem Control Options", 552),
        10, 30,
        /* center line, column */
        i, 30,
        /* form height, width */
        F_NOVERIFY, F_NO_HELP,
        /* Control flags, help message */
        InxMSG("Save Changes?", 585),
        NUTHandle);

    /* Confirm message, handle */
    le /*

    NMSdestroyForm(NUTHandle);

    )

    /*****

    * MainOptionsHandler(void)

    * Description:
    *   This routine is where the main agent thread lives.
    *   It initiates the NUT screen, waits for the DPC MUID
    *   to become active, and wait for user commands.

    * Input:
    *   nothing

    * Output:
    *   nothing

    * Returns:
    *   nothing

    * *****/

    void MainOptionsHandler()
    {
        int choice, prevChoice, i;
        int exitFlag = FALSE;
        LIST *defaultList;
        LONG type;
        BYTE value;
        int countdown = MAX_COUNTDOWN;
        LONG mainPortal;

        #if DRIVER_IO
        LONG removedCount;
        #else
        void (*ControlEntryPoint) () = NULL;
        LONG board;
        struct DriverConfigurationStructure *config;
        char *configName;
        #endif

        /*
        * Clear the screen by creating huge portal with VNORMAL attribute.
        */
        GetScreenSize(&ScreenHeight, &ScreenWidth);
        ScreenHeight = ScreenHeight - NUTHandle->headerHeight;
        mainPortal = NMScreatePortal(
            NUTHandle->headerHeight, 0,
            /* line, column

```



```

(
    CLSGetMLIDControlEntry(board, (void(*)())&ControlEntryP
oInt);
    if (ControlEntryPoint)
    {
        config = (struct DriverConfiguratonStructure *)
            CommandMlidBoar
d, 0, (LONG)ControlEntryPoint);
        if (config)
        {
            configName = config->DshortName;
            if (!CStringComp(configName, DPCName))
                goto FoundDPCBoardNumber;
        }
    }
    delay(500);
    if (NMSKeyStatus(NUTHandle))
    {
        NMSGetKey(&type, &value, NUTHandle);
        if ((type == K_ESCAPE) || (type == K_AF10))
        {
            NMSEndWait(NUTHandle);
            return;
        }
        if (--countdown == 0)
        {
            Spin(NUTHandle);
            countdown = MAX_COUNTDOWN;
        }
    }
}
/*
 * OK. We have a DPC MLID. Lets set up the main menu and
 * wait for the user to do something.
 */
#endif

/* DRIVER_IO
 * FoundDPCBoardNumber:
 */
#endif

NMSEndWait(NUTHandle);
NMSEnableInterruptKey(K_AF10, ExitHandler, NUTHandle);

/*
 * Initialize the main options menu.
 */
NMSInitDhList(NUTHandle, Free); /* Don't sort menu items. */

NMSSetDynamicMessage(DYNAMIC_MESSAGE_ONE,
    (BYTE *)Package Delivery, &NUTHandle->messages);
NMSSetDynamicMessage(DYNAMIC_MESSAGE_TWO,
    MSG("Display MLID Stats", 102), &NUTHandle->messages);
NMSSetDynamicMessage(DYNAMIC_MESSAGE_THREE,
    MSG("DPC Configuration", 95), &NUTHandle->messages);
NMSSetDynamicMessage(DYNAMIC_MESSAGE_FOUR,
    MSG("Dish Pointing", 344), &NUTHandle->messages);
NMSSetDynamicMessage(DYNAMIC_MESSAGE_FIVE,
    MSG("Adapter Information", 150), &NUTHandle->messages);
NMSSetDynamicMessage(DYNAMIC_MESSAGE_SIX,
    MSG("Modem Control", 501), &NUTHandle->messages);
NMSSetDynamicMessage(DYNAMIC_MESSAGE_SEVEN,

if (PackageDelivery)
    NMSAppendToMenu(DYNAMIC_MESSAGE_ONE, 1, NUTHandle);
NMSAppendToMenu(DYNAMIC_MESSAGE_TWO, 2, NUTHandle);
NMSAppendToMenu(DYNAMIC_MESSAGE_THREE, 3, NUTHandle);
NMSAppendToMenu(DYNAMIC_MESSAGE_FOUR, 4, NUTHandle);
NMSAppendToMenu(DYNAMIC_MESSAGE_FIVE, 5, NUTHandle);
NMSAppendToMenu(DYNAMIC_MESSAGE_SIX, 6, NUTHandle);
NMSAppendToMenu(DYNAMIC_MESSAGE_SEVEN, 7, NUTHandle);
while (exitFlag == FALSE)
{
    prevChoice = choice;

    /* Set default list to previous choice */
    defaultList = NMSGetListHead(NUTHandle);
    if (!PackageDelivery)
        choice = 1;
    for (i = choice - 1; i; i--)
        defaultList = defaultList->next;

    choice = NMSMenu(InxMSG("DPCAGENT Options", 151),
        10, 40, defaultList, NULL, NUTHandle, NULL);
    if (removedCount != DIORemovedCount)
    {
        NMSDestroyMenu(NUTHandle);
        NMSClearPortal(NUTHandle->portal[mainPortal]);
        goto LookForAdapter;
    }
    switch (choice) {
    case 1:
        if (NMSPushList(NUTHandle)) {
            DisplayPDInterface();
            NMSPopList(NUTHandle);
        }
        break;
    case 2:
        /* Display MLID Stats */
        if (NMSPushList(NUTHandle)) {
            DisplayMLIDStats();
            NMSPopList(NUTHandle);
        }
        break;
    case 3:
        /* Modem Configuration */
        if (NMSPushList(NUTHandle)) {
            DPCConfiguration();
            NMSPopList(NUTHandle);
        }
        break;
    case 4:
        /* Signal Strength Meter */
        if (NMSPushList(NUTHandle)) {
            DPCPointing();
            NMSPopList(NUTHandle);
        }
        break;
    case 5:
        /* Display Adapter Information */
        if (NMSPushList(NUTHandle)) {
            DisplayAdapterInfo();
            NMSPopList(NUTHandle);
        }
    }
}

```

```

break;
}

case 6:
/* Display Adapter Information */
if (NMSPushLast(NUTHandle)) {
ModemControl();
NMSPopList(NUTHandle);
}
break;

default:
if (NMSConfirm(INMSG("Exit DPCAGENT?", 152), 0, 0, TRUE, NULL,
NUTHandle, NULL) == TRUE)
exitFlag = TRUE;
else
if (choice != 7)
choice = prevChoice;
}
)

NMSDestroyMenu(NUTHandle);
)

.....
DPCAgentMain(void *parm)
.....
Description:
Main thread. It will initialize the NUT screen and wait
user input.

Input:
parm - ignored

Output:
Nothing

Returns:
Nothing

.....
void DPCAgentMain(void *parm)
{
parm = parm;

MainOptionsHandler();
ReturnResources(1);
exit(1);
}

/*****
main(int argc, char *argv[])
Description:
Initialization routine.

Input:
Nothing

```

```

Output:
Nothing

Returns:
0 if successfully initialized
.....

```

```

LONG ScreenID;
LONG main(int argc, char *argv[])
{

```

```

LONG currentScreen;
LONG ser[2];
LONG ccode;
int i;

```

```

for (i = 1; i < argc; i++)
{

```

```

if (ICMPB(argv[i], MSG("-DEBUG", 182), 6) == -1)
{
if ((DebugFlag = strtoul(argv[i][6], 0, 16)) == 0)
DebugFlag = TRUE;
}
}

```

```

/* Get a handle for allocating a resource tag */
NLMHandle = (struct LoadDefinitionStructure *)GetNLMHandle();
if (!NLMHandle)
return(-1);

```

```

if (ReturnMessageInformation((LONG)NLMHandle, (BYTE *)"NLMMessageTabl
e, NULL, NULL, NULL))
return(-1);

```

```

OSGetCountryInfo(&gbldosCountryInfo);

```

```

/* Allocate a resource tag to use for memory allocations */
allocRTag = AllocateResourceTag(NLMHandle, TxtMSG("DPCAGENT Memory", 167
),

```

```

AllocSignature);

```

```

AESTag = AllocateResourceTag(NLMHandle, TxtMSG("DPCAGENT AES", 168),
AESProcessSignature

```

```

169),
asynCIOtag = AllocateResourceTag(NLMHandle, TxtMSG("DPCAGENT AsyncIO",

```

```

ASYNCSIOSignature);

```

```

170),
timerTag = AllocateResourceTag(NLMHandle, TxtMSG("DPCAGENT Delay Timer",

```

```

TimerSignature);

```

```

if (allocRTag == NULL ||
AESTag == NULL ||
timerTag == NULL ||
ASYNCSIOSignature == NULL)

```

```

{
ConsolePrintf(TxtMSG("DPCAGENT: Unable to allocate Resource Tags
", 171));
return(-1);
}

```

```

/* Create a screen for displaying our information */

```

```

currentScreen = GetCurrentScreen();
SetAutoScreenDestructionMode(TRUE);

/* Initialize the screen interface */
ScreenID = CreateScreen("DPCAgent Utility", AUTO_DESTROY_SCREEN);
ccode = NWSInitializeNut(InxMSG("DPC AGENT PROGRAM", 172), AGENT_VERSION);

SMALL_HEADER, NUT_REVISION_LEVEL, NULL, NULL,
ScreenID, (LONG)allcrttag, &NUTHandle);
if (ccode)
{
    ConsolePrintf(TEXTMSG("DPCAGENT: Unable to initialize NUT.", 174));
    return(-1);
}

NUMMessageTable = (BYTE **)&(NUTHandle->messages);

/* Get a connection with the server we're on */
if (DebugFlag) {
    DisplayScreen(ScreenID);
    SetCurrentScreen(currentScreen);
}

#define LOG_ECB_ACTIVITY
if (DebugFlag >= 0x51) {
    DPC_TGID = GetThreadGroupID();
    LogRegisterClient("SYS:DIRECTPC/LOG_CFG", 2048, &LogClientHandle);
    LogRegisterEvent("ECB", &LogECBHandle);
}

#endif /* LOG_ECB_ACTIVITY */
else {
    DestroyScreen(currentScreen);
}

#ifdef DEBUG_ALL
if (Openscreen(MSG("DPCAGENT Debug Screen", 224), screenTag, &DebugScreenID))
{
    ConsolePrintf(MSG("DPCAGENT: Unable to create debug screen.", 225));
    return(-1);
}
#endif

DPCUpdateConfig();

InetChangeProtocol();

DPCSetMaxConnections(ser);

DPCAgentPID = BeginThread(DPCAgentMain, NULL, NULL, NULL);
RenameThread(DPCAgentPID, "DPCAgent Main");
if (PackagedDelivery) {
    DPCFilePID = BeginThread(DPCFileMain, NULL, 32 * 1024, NULL);
    RenameThread(DPCFilePID, "DPCAgent PD");
    DPCAccessPID = BeginThread(DPCAccessMain, NULL, NULL, NULL);
    RenameThread(DPCAccessPID, "DPCAgent Access");
}
DPCModemPID = BeginThread(DPCModemMain, NULL, NULL, NULL);
RenameThread(DPCModemPID, "DPCAgent Modem");
if (DPCMaxConnections) {
    DPCInetPID = BeginThread(InetMain, NULL, NULL, NULL);
    RenameThread(DPCInetPID, "DPCAgent Tinet");
}
signal(SIGTERM, ReturnResources);
ExitThread(TSR_THREAD, 0);
}

ReturnResources(int sig)
{
    Description:
        Shutdown routine. Returns the modules resources.
    Input:
        sig
        - ignored
    Output:
        Nothing
    Returns:
        Nothing
}

void ReturnResources(int sig)
{
    int i;
    int countdown = MAX_COUNTDOWN;

    sig = sig;
    ExitingFlag = TRUE;
    if (InReturnResources)
        return;

    InReturnResources = TRUE;

    /*
    * Force NUT to escape out of all menus so that it can clean
    * before we call NWSRestoreNUT(NUT has a bug where it will
    * attempt to free up its memory twice(ABEND) if the user
    * leaves the screen a couple of menus in before unloading
    * the application from the command line.
    */

    for(i = 0; i < 4; i++)
        NWSUngetkey(UGK_ESCAPE_KEY, UGK_NORMAL_KEY, NUTHandle);

    StartWaitWithMessage(ScreenHeight/2, ScreenWidth/2, NUTHandle,
        InxMSG("Waiting for threads to Exit", 226));

    if (AccessAsleep)
        ResumeThread(DPCAccessPID);

    if (InetAsleep)
        ResumeThread(DPCInetPID);

    /*
    * Give threads and NUT a chance to execute.
    * Wait 1.5 seconds since signal meter and stats could be sleeping
    * for up to a second.
    */
    delay(1500);

    while(DPCFilePID || DPCModemPID || DPCAccessPID || DPCInetPID) {
        void DPCPterminate(void);
        DPCPterminate();
        if (AccessAsleep)
            ResumeThread(DPCAccessPID);
    }
}

```

```
    if (InetAsleep)
        ResumeThread(DPCInetPID);

    if (--countdown == 0) {
        Spin(NUTHandle);
        countdown = MAX_COUNTDOWN;
    }
    ThreadSwitchWithDelay();
}

#ifdef LOG_ECB_ACTIVITY
    if (DebugFlag >= 0x51) {
        LogDerRegisterEvent(&LogECBHandle);
        LogDerRegisterClient(&LogClientHandle);
    }
#endif /* LOG_ECB_ACTIVITY */

    if (DIOCfg.out_protocol == OUT_PPP)
    {
        DisconnectPPP();
    }
    DIODeRegisterAgent();

#ifdef DEBUG_ALL
    CloseScreen(DebugScreenID);
#endif
    NWSendWait(NUTHandle);
    NWSRestoreNut(NUTHandle);
}
```

```

#include "dpcagent.h"          /* Our header file */

LONG
int
/*
 * Access Configuration Variables
 */
BYTE SiteID[9];
WORD CDBVersion = 0;
WORD CDBethVersion = 0;
BYTE DcauFlag = 0;
LONG DcauTime = 0;
DCAUrequest_t DcauRequest;
CASDBuffer CASDBpacau;
CASDBuffer CASDBpbeb;
CASDBuffer CASDBbecau;

/* Elements tables */
static CDBelement_t Elements[MAXELEMENTS];
static CDBelement_t UpdatedElements[MAXELEMENTS];

BYTE *RecvBuf;
MACrecord_t CASmacs[MAX_MAC_RECORDS];
BYTE AccessAddress[] = {0x03, 0x00, 0x00, 0x00, 0x00, 0x00};
LONG AccessSleep = FALSE; /* Access thread needs t
o wake up flag */

BYTE
/*
 * Stores current packet
 */
/*
 * ECB linked list variables.
 */
static ECB *AccessECBHead = 0; /* Take ECBS from here */
static ECB *AccessECBTail = 0; /* Put ECBS here */

/*
 * element key used by LroSetAddress.
 */
BYTE MagicKey[] = { 0x11, 0x11, 0x11, 0x11, 0x11, 0x11, 0x11 };

/*
 * Address Type Table used by MACbuildAddr().
 */
static unsigned char table[5][2] = {
/*
 * Hybrid Internet */
/*
 * CAS individual */
/*
 * Package Delivery */
/*
 * Data Feed */
/*
 * BYPASS */
/*
 * indiv/mult
 * MAC_INDIVID, MAC_NORMAL },
 * MAC_INDIVID, MAC_BYPASS },
 * MAC_MULTICAST, MAC_NORMAL },
 * MAC_MULTICAST, MAC_NORMAL },
 * MAC_MULTICAST, MAC_BYPASS }
};

BYTE SerialNum[9];
BYTE SerialNumPacked[3];

/*
 * ReadConfig(void)
 */
/*
 * Description:
 * This routine reads the DPC.CFG file and stores the conte
nts
 */

```

into the appropriate globale variables.

```

*
* Input:
*   Nothing
*
* Output:
*   Nothing
*
* Returns:
*   Nothing
*
*****
static void ReadConfig(void)
{
    int handle, k;
    BYTE *mem_ptr;

    for(k = 0; k < MAXELEMENTS; k++)
    {
        Elements[k].in_use = 'N';
        UpdatedElements[k].in_use = 'N';
    }

    handle = open(MSG("SYS:DIREPC\\DB\\DPC.CFG", 203), O_RDONLY);
    if (handle != -1)
    {
        read(handle, &SiteID, sizeof(SiteID));
        read(handle, &CDBVersion, sizeof(CDBVersion));
        read(handle, &CDBethVersion, sizeof(CDBethVersion));
        read(handle, &DcauFlag, sizeof(DcauFlag));
        read(handle, &DcauTime, sizeof(DcauTime));
        read(handle, &DcauRequest, sizeof(DCAUrequest_t));
        read(handle, &CASDBpacau, sizeof(CASDBbuffer));
        read(handle, &CASDBpbeb, sizeof(CASDBbuffer));
        read(handle, &CASDBbecau, sizeof(CASDBbuffer));
        read(handle, &CASDBbecau, sizeof(CASDBbuffer));

    }
    else
        UpdateFileStatus(MSG("Obtaining Encryption Keys", 204));

    for(k = 0; k < MAX_MAC_RECORDS; k++)
        CASmacs[k].in_use = 'N';

    if (handle != -1)
    {
        close(handle);
    }
    else
    {
        SiteID[0] = '\0';
        CASDBpacau.version = CASDBpbeb.version = 0;
        CASDBbecau.version = CASDBpbeb.version = 0;
        CASDBpacau.entries = CASDBpbeb.entries = 0;
        CASDBbecau.entries = CASDBpbeb.entries = 0;
    }

    CASDBpacau.entry_len = PACAU_LEN;
    CASDBpbeb.entry_len = PEB_LEN;
    CASDBbecau.entry_len = DCAU_LEN;
    CASDBbecau.entry_len = BCAU_LEN;

    if (DcauFlag)
    {
        UpdateFileStatus(MSG("Retry: Obtaining Encryption Keys", 257));
    }
}

```

```
DacauRequest.opcode = DACAU_REQUEST;
RandDacauTime = time(0) + rand();
```

```
SaveConfig(void *parm)
```

```
Description:
```

```
This routine writes the access structures out to DPC.CFG
a change is made to any of the structures.
```

```
Input:
```

```
Nothing
```

```
Output:
```

```
Nothing
```

```
Returns:
```

```
Nothing
```

```
static void SaveConfig(void)
```

```
int handle;
```

```
int tmpFlag;
```

```
handle = open(MSC("SYS:DIRECPC\\DB\\DPC.CFG", 206), O_RDWR | O_CREAT, S_
```

```
WRITE | S_READ);
```

```
tmpFlag = DacauFlag;
```

```
if (WaitingForKeys)
```

```
{
    DacauFlag = 1;
```

```
}
```

```
/* Write version */
```

```
write(handle, SiteID, sizeof(SiteID));
```

```
write(handle, &CDBVersion, sizeof(CDBVersion));
```

```
write(handle, &CDBEthVersion, sizeof(CDBEthVersion));
```

```
write(handle, &DacauFlag, sizeof(DacauFlag));
```

```
write(handle, &DacauTime, sizeof(DacauTime));
```

```
write(handle, &DacauRequest, sizeof(DACAUrequest_t));
```

```
/* Write info headers */
```

```
write(handle, &CASDBpacau, sizeof(CASDBbuffer));
```

```
write(handle, &CASDBpeb, sizeof(CASDBbuffer));
```

```
write(handle, &CASDBdaca, sizeof(CASDBbuffer));
```

```
write(handle, &CASDBbecau, sizeof(CASDBbuffer));
```

```
/* Write buffers */
```

```
close(handle);
```

```
DacauFlag = tmpFlag;
```

```
AccessESR(ECB *ecb)
```

```
Description:
```

```
This routine is called by the MUID interrupt service
routine when a packet is received. The ECB is queued
and if the Access file thread is sleeping, it is
```

```
woken up.
```

```
Input:
```

```
ecb
```

```
the packet
```

```
Output:
```

```
nothing
```

```
Returns:
```

```
0
```

```
We always keep the ECB
```

```
int AccessESR(ECB *ecb)
```

```
/* Link the ECB to the Tail of the linked list */
ecb->ECB_Nextlink = 0;
```

```
if (AccessECBTail)
```

```
AccessECBTail->ECB_Nextlink = ecb;
```

```
AccessECBTail = ecb;
```

```
if (AccessECBHead == 0)
```

```
AccessECBHead = ecb;
```

```
/* Wake up file thread only if we need to */
if (AccessAsleep)
```

```
ResumeThread(DPCAccessPID);
```

```
#ifdef LOG_ECB_ACTIVITY
```

```
if (LogECBHandle) {
```

```
int TGID = SetThreadGroupID(DPC_TGID);
```

```
LogMsg(LogClientHandle, LogECBHandle, FALSE,
```

```
"ACCESS queue(%08lx)\n", ecb);
```

```
SetThreadGroupID(TGID);
```

```
#endif /* LOG_ECB_ACTIVITY */
```

```
return(0);
```

```
.....
```

```
find_pebID element_pattern,
```

```
char ver_pattern)
```

```
Description:
```

```
This routine searches the PEB list to find an entry that
matches the element and version pattern passed in.
```

```
Input:
```

```
element_pattern
```

```
ver_pattern
```

```
to search for
```

```
Output:
```

```
nothing
```

```
Returns:
```

```
pointer to peb entry if it was found
otherwise its a NULL
```

```
.....
```

```
MUXecau_t *find_ecau(ID group_pattern, char ver_pattern)
```

```

int i;
MUXecau_t *p_ecau, *ret = NULL;

for(i = 0; i < CASDBecau.length; i += ECAU_LEN)
{
    p_ecau = (MUXecau_t *) (CASDBecau.p_buffer + i);
    if(CmpB(&p_ecau->groupid, &group_pattern, sizeof(ID)) == -1)
    {
        if(ver_pattern == -1)
        {
            ret = p_ecau;
            break;
        }
        else
        {
            if(ver_pattern == p_ecau->version)
            {
                ret = p_ecau;
                break;
            }
        }
    }
    return(ret);
}

static MUXpeb_t *find_peb(ID element_pattern, char ver_pattern)
{
    unsigned short i, num_addr;
    MUXpeb_t *p_peb, *ret = NULL;

    for(i = 0; i < CASDBpeb.length; i += PEB_LEN)
    {
        p_peb = (MUXpeb_t *) (CASDBpeb.p_buffer + i);
        if(CmpB(&p_peb->elementid, &element_pattern, sizeof(ID)) == -1)
        {
            if(ver_pattern == -1)
            {
                ret = p_peb;
                break;
            }
            else
            {
                if(ver_pattern == p_peb->version)
                {
                    ret = p_peb;
                    break;
                }
            }
        }
    }

    num_addr = p_peb->num_addr(0);
    num_addr |= ((unsigned short) p_peb->num_addr[1]) << 8;
    i += num_addr * MAC_LENGTH;

    return(ret);
}

/*
 * Deletes element not only from the CASDB
 * but from adapter as well
 */
.....
del_peb_element(int e_num)
.....
}

Description:
    This routine deletes and from the CASDB and from the
    adapter.

Input:
    e_num          - index of the element to
                    delete

Output:
    nothing

Returns:
    0

.....
static del_peb_element(int e_num)
{
    int k;

    DIDeleteAddress(Elements[e_num].channel, (BYTE *) &Elements[e_num].e_mac);

    for(k = 0; k < MAX_MAC_RECORDS; k++)
    {
        if(CASmacs[k].in_use == 'Y' &&
           CmpB(&CASmacs[k].dpc_mac, &Elements[e_num].e_mac, sizeof(MACadd
r_t)) == -1)
        {
            CASmacs[k].in_use = 'N';
            break;
        }
    }
    Elements[e_num].in_use = 'N';
    return(0);
}

.....
replace_peb_element(int num,
                    unsigned char new_ver)
.....
Description:
    This routine replaces the version numbers of the element
    indexed by num.

Input:
    int num
    nt to modify    new_ver
    stuff into Element entry

Output:
    nothing

Returns:
    0

.....
static replace_peb_element(int num, unsigned char new_ver)
{
    int k;
}

```

```

    for(k = 0; k < MAX_MAC_RECORDS; k++)
    {
        if(CASmacs[k].in_use == 'Y' &&
           CcMPB(&CASmacs[k].dpc_mac, &Elements[num].e_mac, sizeof(MACaddr_
c)) == -1)
        {
            CASmacs[k].dpc_mac.Ver = new_ver;
            break;
        }
        Elements[num].e_mac.Ver = new_ver;
        Elements[num].e_ver = new_ver;
        return 0;
    }
}

/*****
 *
 * find_pacau(ID group_pattern,
 *            char ver_pattern)
 *
 * Description:
 *   This routine attempts to locate a pacau given a group
 *   pattern.
 *
 * Input:
 *   group_pattern
 *   ver_pattern
 *
 * - group pattern to match
 * - version portion of the
 * pattern
 *
 * Output:
 *
 * Returns:
 *   pointer to pacau if successful
 *   Otherwise NULL
 *
 * .....
```

```

MUXpacau_t *find_pacau(ID group_pattern, char ver_pattern)
{
    int i;
    MUXpacau_t *p_pacau, *ret = NULL;
    for(i = 0; i < CASDBpacau.length; i += PACAU_LEN)
    {
        p_pacau = (MUXpacau_t *) (CASDBpacau.p_buffer + i);
        if((CcMPB(&p_pacau->groupid, &group_pattern, 3)) == -1)
        {
            if(ver_pattern == -1)
            {
                ret = p_pacau;
                break;
            }
            else
            {
                if(ver_pattern == p_pacau->version)
                {
                    ret = p_pacau;
                    break;
                }
            }
        }
        return(ret);
    }
}

/*****
 *
 * find_dacau(ID group_pattern,
 *            char ver_pattern)
 *
 * Description:
 *   This routine attempts to locate a dacau given a group
 *   pattern.
 *
 * Input:
 *   group_pattern
 *   ver_pattern
 *
 * - group pattern to match
 * - version portion of the
 * pattern
 *
 * Output:
 *
 * Returns:
 *   pointer to dacau if successful
 *   Otherwise NULL
 *
 * .....
```

```

MUXdacau_t *find_dacau(ID group_pattern, char ver_pattern)
{
    int i;
    MUXdacau_t *p_dacau, *ret = NULL;
    for(i = 0; i < CASDBdacau.length; i += DACAU_LEN)
    {
        p_dacau = (MUXdacau_t *) (CASDBdacau.p_buffer + i);
        if((CcMPB(&p_dacau->groupid, &group_pattern, 3)) == -1)
        {
            if(ver_pattern == -1)
            {
                break;
            }
            else
            {
                if(ver_pattern == p_dacau->version)
                {
                    ret = p_dacau;
                    break;
                }
            }
        }
        return(ret);
    }
}

/*****
 *
 * reverse_key(BYTE *key)
 *
 * Description:
 *   This routine swaps the byte order of the 8 byte
 *   key passed in.
 *
 * Input:
 *   key
 *
 * - key to reverse
 *
 * Output:
 *
 * Returns:
 *   pointer to pacau if successful
 *   Otherwise NULL
 *
 * .....
```



```
void reverse_key(BYTE *key)
```

```
{
    unsigned char x;
```

```
    x = key[0]; key[0] = key[1]; key[1] = x;
    x = key[2]; key[2] = key[3]; key[3] = x;
    x = key[4]; key[4] = key[5]; key[5] = x;
    x = key[6]; key[6] = key[7]; key[7] = x;
}
```

```
make_element_id(BYTE *e_id,
                char *e_id_txt)
```

```
Description:
```

```
This routine is called by lrosetcaddress to help
build the element id.
```

```
Input:
```

```
    e_id
```

```
- element id
```

```
Output:
```

```
    e_id_txt
```

```
- element text
```

```
Returns:
```

```
    nothing
```

```
void make_element_id(BYTE *e_id, char *e_id_txt)
```

```
{
    BYTE work[3];
```

```
    static char HexChar[] = MSG("0123456789ABCDEF", 208);
```

```
    unsigned char Ch, *p;
    int count = 3, i = 0;
```

```
    work[0] = e_id[2];
    work[1] = e_id[1];
    work[2] = e_id[0];
```

```
    p = work;
```

```
    while (count--)
```

```
    {
        Ch = *p++;
```

```
        e_id_txt[i++] = HexChar[Ch>>4]; /* high nibble */
```

```
        e_id_txt[i++] = HexChar[Ch & 0x0F]; /* low nibble */
```

```
    }
    e_id_txt[i] = '\0';
```

```
int43(LONG id, BYTE *array)
```

```
Description:
```

```
This routine is called by MACbuildaddr to convert
an id to its 3 byte equivalent.
```

```
Input:
```

```
    id
```

```
- id to convert
```

```
array
```

```
r to 3 byte array
```

```
Output:
```

```
    array is filled in
```

```
Returns:
```

```
    0 if successful
```

```
static int43(LONG id, BYTE *array)
```

```
{
    union {
```

```
        BYTE b[4];
```

```
        LONG w;
```

```
    } offset;
```

```
    int status = 0;
```

```
    offset.w = id;
```

```
    if (offset.b[3]==0 && !(offset.b[2] & 0xc0)) {
```

```
        array[0] = offset.b[2];
```

```
        array[1] = offset.b[1];
```

```
        array[2] = offset.b[0];
```

```
    }
```

```
    else {
```

```
        array[0]=array[1]=array[2] = 0;
```

```
        status = -1;
```

```
    }
```

```
    return status;
```

```
MACbuildaddr(char *element_txt,
```

```
            int feature,
```

```
            BYTE ver,
```

```
            MACaddr_t *address)
```

```
Description:
```

```
This routine is called by lrosetcaddress to build a
MAC address out of a file_id and community id.
```

```
Input:
```

```
    element_txt
```

```
    feature
```

```
    AC_PKG, MAC_DF
```

```
    _BYPASS_MULTICAST
```

```
    community id
```

```
    address
```

```
    ing address
```

```
Output:
```

```
    address is filled in
```

```
Returns:
```

```
    0 if successful
```

```
int MACbuildaddr(char *element_txt, int feature, BYTE ver, MACaddr_t *address)
```

```
{
    BYTE element[3];
```

```
    LONG i;
```

- pointer

```

int k, status = 0;

if(feature < 0 || feature > 5)
    return(-1);
/* Step one */
scanf(element_txt, MSG("%ix", 137), &i);
if(status == INT43(1, element)) != 0)
    return(status);
/* Step two */
for(k=0; k<3; k++)
    element[k] = element[k] << 2;
element[k] |= ((k == 2) ? 0x00 : element[k+1]) >> 6;

/* Step three */
address->Element(0) = element(2);
address->Element(1) = element(1);
address->Element(2) = element(0);
/* Set Multicast/Individual */
if(table[feature][0] == MAC_MULTICAST)
    address->Element(0) |= MAC_MULTICAST;
/* Set Bypass/Normal */
if(table[feature][1] == MAC_BYPASS)
    address->Element(0) |= MAC_BYPASS;
/* Set Application ID or Version number */
switch(feature)
{
    case MAC_HI:
        address->Ver = 0x02;
        break;
    case MAC_CAS_IND:
        address->Ver = 0x01;
        break;
    case MAC_BYPASS_MULTICAST:
        address->Ver = 0x00;
        break;
    default:
        address->Ver = ver;
        break;
}
/* Set reserved field */
address->Reserved(0) = address->Reserved(1) = 0x00;
if(feature == MAC_DF)
    address->Element(2) = 0xff;
return(status);
}

/*.....*/
find_peb_mac(MACaddr_t mac_pattern)
{
    Description:    Finds the PEB entry which contains the mac address passed in.

    Input:          mac_pattern

    Output:         Nothing

    Returns:        NULL if no entry found
                   Otherwise its a pointer to the PEB entry
}

```

```

static MUXpeb_t *find_peb_mac(MACaddr_t mac_pattern)
{
    unsigned short i, j, num_addr;
    MUXpeb_t *p_peb, *ret = NULL;
    for(i = 0; i < CASDBpeb.length; i += PEB_LEN)
    {
        p_peb = (MUXpeb_t *) (CASDBpeb.p_buffer + i);
        num_addr = p_peb->num_addr(0);
        num_addr |= ((unsigned short) p_peb->num_addr(1)) << 8;
        for(j = 0; j < num_addr; j++)
        {
            if(CMPB(&mac_pattern,
                (unsigned char *) p_peb + PEB_LEN + j * MAC_LENGTH, MAC_LENGTH) == -1)
            {
                ret = p_peb;
                goto peb_mac_exit;
            }
            i += num_addr * MAC_LENGTH;
        }
        peb_mac_exit:
        return(ret);
    }
    /*.....*/
    find_element_id(ID id, BYTE ver)
    {
        Description:    Find the elements index into the Element table.

        Input:         id
                     ver
                     - 1 byte version

        Output:        Nothing

        Returns:       -1 if not found
                     otherwise its the index
    }
    /*.....*/
    static find_element_id(ID id, BYTE ver)
    {
        int k;
        int ret = -1;
        for(k = 0; k < MAXELEMENTS; k++)
        {
            if(Elements[k].in_use == 'Y' &&
                CMPB(Elements[k].e_id, &id, sizeof(ID)) == -1 &&
                Elements[k].e_ver == ver)
            {
                ret = k;
                break;
            }
        }
    }
}

```

```
return ret;
```

```
hextoi(int c)
```

```
Description:
```

```
Converts an ASCII hex character into an integer.
```

```
Input:
```

```
c
```

```
- ASCII
```

```
Output:
```

```
Nothing
```

```
Returns:
```

```
-1 if not hex character  
otherwise its the integer equivalent
```

```
static int hextoi(  
int c)
```

```
char digit, lower, upper;
```

```
digit = (c >= '0' && c <= '9');
```

```
lower = (c >= 'a' && c <= 'f');
```

```
upper = (c >= 'A' && c <= 'F');
```

```
if (digit)
```

```
return (c - '0');
```

```
if (lower)
```

```
return (c - 'a' + 10);
```

```
if (upper)
```

```
return (c - 'A' + 10);
```

```
return (-1);
```

```
pack_mac_addr(BYTE *packed_address,  
int packed_address_len,  
BYTE *address,  
int address_len)
```

```
Description:
```

```
Converts a character string containing the mac address into  
packed BCD digits.
```

```
Input:
```

```
packed_address_len
```

```
address
```

```
invert
```

```
I string
```

```
Output:
```

```
packed_buffer
```

```
into
```

```
Returns:
```

```
TRUE if packed successfully
```

```
#define LEFT 0
```

```
#define RIGHT 1
```

```
int pack_mac_addr(BYTE *packed_address, int packed_address_len,  
BYTE *address, int address_len)
```

```
BYTE c, side;
```

```
int i, j;
```

```
/*
```

```
* Pack hex digit ascii string in "address" into binary in  
* "packed_address". Return FALSE if unsuccessful. If number of hex  
* digits in "address" cannot fill "packed_address", then  
* "packed_address" is padded to the right with zeros.  
*/
```

```
side = LEFT;
```

```
for (i = 0; i < packed_address_len; i++)
```

```
packed_address[i] = 0;
```

```
for (i = 0, j = 0; i < address_len; i++)
```

```
if ((c = hextoi(address[i])) == 0xff)
```

```
return (FALSE);
```

```
if (side == LEFT)
```

```
{
```

```
c = c << 4;
```

```
}  
packed_address[j] |= c;
```

```
if (++side > RIGHT)
```

```
{
```

```
side = LEFT;
```

```
if (++j > packed_address_len)
```

```
return (FALSE);
```

```
}  
return (TRUE);
```

```
check_df_groups(void)
```

```
Description:
```

```
This function is called when new PACAU or DACAU is proce
```

```
ssing.
```

```
It checks if there are any changes in groups membership
```

```
to avoid receiving DF elements when membership of this a
```

```
to the proper DF group has been deleted.
```

```
Input:
```

```
Nothing
```

```
Output:
```

```
Nothing
```

```
Returns:
```

```
0 if check was successful
```

```

static check_df_groups(void)
{
    int i;
    MUXpacau_c *pacau;
    MUXdacauc_c *dacauc;
    MUXpeb_c *p_peb;

    for(i = 0; i < MAXELEMENTS; i++) {
        if(Elements[i].in_use == 'N' || Elements[i].packfeed != 'F')
            continue;
        if(p_peb = find_peb(Elements[i].e_id, -1)) == NULL
            continue;
        pacau = find_pacau(p_peb->groupid, p_peb->version);
        dacauc = find_dacauc(p_peb->groupid, p_peb->version);

        if(dacauc == NULL && pacau == NULL) {
            /* We have no group for this element */
            del_peb_element(i);
        }
        return 0;
    }

    /*.....*/

    parse_pacau(BYTE *buf, WORD len)

    Description:
        Parse the PACAU packet. This is where we detect that we
        must receive a new key via the modem(packets d version
        will not match CASDBdacauc.version).

    Input:
        buf          - pointer to the message receive
        len          - length of the message received

    Output:
        Nothing

    Returns:
        0 if successfully parsed
        .....*/

static parse_pacau(BYTE *buf, WORD len)
{
    LONG curr_p_version;
    LONG curr_d_version;
    LONG curr_d_time;
    int ret = 0, k;

    if(strlen(buf, SITEID, 8) != ESUCCESS) {
        strcpy(SITEID, buf, 8);
        SITEID[8] = 0;
        CDBversion++;
        /* New Site ID - reset versions of PACAU, DCAUC, ...*/
        CASDBpacau.version = CASDBpeb.version = 0;
        CASDBdacauc.version = CASDBdacauc.version = 0;
        SaveConfig();
    }

    curr_p_version =
        buf[8] + 256UL +
        buf[9] * 256UL +
        buf[10] * 256UL +
        buf[11] * 256UL +
        256UL;

    curr_d_version =
        buf[12] +
        buf[13] * 256UL +
        buf[14] * 256UL +
        buf[15] * 256UL +
        256UL;

    curr_d_time =
        buf[16] +
        buf[17] * 256UL +
        buf[18] * 256UL +
        buf[19] * 256UL +
        256UL;

    if(curr_d_version != CASDBdacauc.version) {
        /* There are some changes in the DCAUC staff */
        /* Build DCAUC request */
        Dcaucflag = 1;
        srand(time(0));
        DcaucTime = curr_d_time;
        RndDcaucTime = time(0) + rand();
        CDBversion++;

        WaitingForKeys = TRUE;
        UpdateFileStatus(MSG("Obtaining Encryption Keys", 138));

        memcpy(&DcaucRequest.d_version, buf + 8 + sizeof(VER), sizeof(VER));
        #ifdef USE_NEW_MIPS_CODE
        DcaucRequest.d_version.i[3] |= 0x80;
        #endif

        CASDBdacauc.version = curr_d_version;
        SaveConfig();
    }

    if(curr_p_version != CASDBpacau.version) {
        CDBversion++;
        CASDBpacau.version = curr_p_version;
        CASDBpacau.length = len - PACAU_HEAD_LEN;
        memcpy(CASDBpacau.p_buffer,
            buf + PACAU_HEAD_LEN,
            CASDBpacau.length);
        CASDBpacau.entries = CASDBpacau.length / CASDBpacau.entry_len;
        check_df_groups();
        SaveConfig();
    }

    return ret;
}

/*.....*/

    parse_dacauc(BYTE *buf, WORD len)

    Description:
        Parse the DCAUC packet. Replace the old entry by the
        new one as long as version doesn't match CASDBdacauc.vera

    Input:
        buf          - pointer to the message receive
        len          - length of the message received

```

```

*
* Output: Nothing
*
* Returns: 0 if successfully parsed
*
*...../

static parse_ecau(BYTE *buf, WORD len)
{
    LONG curr_e_version;
    int ret = 0;

    curr_e_version =
        buf[1] * 256UL +
        buf[2] * 256UL +
        buf[3] * 256UL + 256UL;

    if (curr_e_version != CASDBecau.version)
    {
        CDBVersion++;
        CASDBecau.version = curr_e_version;
        CMovB(buf + ECAU_HEAD_LEN, CASDBecau.p_buffer, len - ECAU_HEAD_LEN);
        CASDBecau.length = len - ECAU_HEAD_LEN;
        CASDBecau.entries = CASDBecau.entry_len;
        SaveConfig();
    }
    return ret;
}

/...../
*
* parse_pdb(BYTE *buf, WORD len)
*
* Description:
*
*   Parse the PEB packet. Replace the old entry by the
*   new one. Check elements and add new addresses to the
*   MUID and delete old ones.
*
* Input:
*
*   buf          - pointer to the message receive
*   len          - length of the message received
*
* Output:
*
*   Nothing
*
* Returns:
*
*   0 if successfully parsed
*
*...../

static parse_pdb(BYTE *buf, WORD len)
{
    int i, k, macs, not_found;
    MUXpbe_t *p_pdb;
    unsigned long curr_version;
    int ret = 0;
    MUXpacau_t *pacau;
    MUXdacau_t *dacau;

    curr_version =
        buf[0] +
        buf[1] * 256UL +
        buf[2] * 256UL + 256UL +

    elements[i].e_mac);

    /* Replace element in the CDB */
    replace_pdb_element(i, p_pdb->version);
    /* Trying to add address */
    /* At first find group for the element */
    pacau = find_pacau(p_pdb->groupid, p_pdb->version);
    dacau = find_dacau(p_pdb->groupid, p_pdb->version);

    if (dacau != NULL)
        pacau = (MUXpacau_t *)dacau;
    if (pacau != NULL)
    {
        if (DIOAddGroupAddress(elements[i].channel, (BYTE *) &pacau->g
    )
    }

    /* Find a group, Add */
    channelCfg.cfgChannel = elements[i].chan
nel;
    channelCfg.cfgNumAddresses = 1;
    CMovB(&elements[i].e_mac, channelCfg.cfg
CMovB(&pacau->g_key, key, 8);
    reverse_key(key);
    CMovB(key, channelCfg.cfgGroupKey, 8);
    CMovB(&magicKey, channelCfg.cfgElementKe
ecb.ECB_StackID = MUID.ADDRESS;
ecb.ECB_Fragment[0].FragmentAddress = &c
del_pdb_element(i);
}

```

```

    else
    {
        /* Can't find group for this element */
        /* We are not subscribed on this group any more */
        /* Delete element. */
        del_pdb_element(i);
    }

    /* Check Ethernet addresses within Element. */
    macs = not_found = 0;
    for(k = 0; k < MAX_MAC_RECORDS; k++)
    {
        if(CASmacs[k].in_use == 'Y' &&
           CCmpB(&CASmacs[k].dpc_mac, &Elements[i].e_mac, s
sizeof(MACaddr_t)) == -1)
        {
            macs++;
            if(!find_pdb_mac(CASmacs[k].e_mac) == NULL)
            {
                /* Somebody in the NOC has deleted Ethernet */
                /* address of this element... */
                not_found++;
                CASmacs[k].in_use = 'N';
            }
        }
        if(macs == not_found)
        {
            /* There are no ethernet addresses for the element */
            del_pdb_element(i);
        }
    }

    SaveConfig();
}
return ret;
}

/*****
 *
 * parse_gup(BYTE *buf, WORD len)
 *
 * Description:
 *   Parse the GUP packet. Add new entries to the PACAU buffer.
 *
 * Input:
 *   buf          - pointer to the message receive
 *   len          - length of the message received
 *
 * Output:
 *   Nothing
 *
 * Returns:
 *   0 if successfully parsed
 *
 *****/
static parse_gup(BYTE *buf, WORD len)
{
    GUPid_t *p_gup_element;
    MUXpacau_t *p_gup_head;
    MUXpacau_t *p_pacau_tmp;
    int i, curr_len = 0;

    p_gup_head = (GUPhead_t *)buf;
    index = CCmpB(&p_gup_head->adapterstart, SerialNum, sizeof(ID));
    if (index == -1)
        start = 0;
    else
        start = p_gup_head->adapterstart.(index) - SerialNum(i);
    index = CCmpB(&p_gup_head->adapterend, SerialNum, sizeof(ID));
    if (index == -1)
        end = 0;
    else
        end = p_gup_head->adapterend.(index) - SerialNum(i);

    if(start <= 0 && end >= 0)
    {
        CDBVersion++;
        for(i = 0; i < p_gup_head->entries && curr_len < len; i++, len -= GUP_LEN)
        {
            p_gup_element = (GUPid_t *) (buf + GUP_HEAD_LEN + i * GUP_LEN);
            if(CCmpB(&p_gup_element->adapternum, SerialNum, sizeof(ID)) == -1)
            {
                p_pacau_tmp = (MUXpacau_t *) (CASDBpacau.p_buffer + CASDBpacau.length);
                CASDBpacau.length += PACAU_LEN;
                CASDBpacau.entries++;
                CMovB(&p_gup_head->groupid, &p_pacau_tmp->groupid, sizeof(ID));
                p_pacau_tmp->version = p_gup_head->g_ver;
                CMovB(&p_gup_element->g_key, &p_pacau_tmp->g_key, sizeof(chunk));
            }
        }
        return 0;
    }

    /*****
     *
     * update_pdb(BYTE *buf, WORD len)
     *
     * Description:
     *   Parse the UPDATE_PDB packet. Replace the old entry with
     *   new one according to the Element ID.
     *
     * Input:
     *   buf          - pointer to the message receive
     *   len          - length of the message received
     *
     * Output:
     *   Nothing
     *
     * Returns:
     *   0 if successfully parsed
     *
     *****/
    static update_pdb(BYTE *buf, WORD len)
    {
        int m;
        MUXpacau_t *pacau = NULL;
        MUXpacau_t *dcau = NULL;

```

```

/*
 * Copyright (c) 1997, 1998, 1999, 2000, 2001, 2002, 2003, 2004, 2005, 2006, 2007, 2008, 2009, 2010, 2011, 2012, 2013, 2014, 2015, 2016, 2017, 2018, 2019, 2020, 2021, 2022, 2023, 2024, 2025, 2026, 2027, 2028, 2029, 2030, 2031, 2032, 2033, 2034, 2035, 2036, 2037, 2038, 2039, 2040, 2041, 2042, 2043, 2044, 2045, 2046, 2047, 2048, 2049, 2050, 2051, 2052, 2053, 2054, 2055, 2056, 2057, 2058, 2059, 2060, 2061, 2062, 2063, 2064, 2065, 2066, 2067, 2068, 2069, 2070, 2071, 2072, 2073, 2074, 2075, 2076, 2077, 2078, 2079, 2080, 2081, 2082, 2083, 2084, 2085, 2086, 2087, 2088, 2089, 2090, 2091, 2092, 2093, 2094, 2095, 2096, 2097, 2098, 2099, 2100, 2101, 2102, 2103, 2104, 2105, 2106, 2107, 2108, 2109, 2110, 2111, 2112, 2113, 2114, 2115, 2116, 2117, 2118, 2119, 2120, 2121, 2122, 2123, 2124, 2125, 2126, 2127, 2128, 2129, 2130, 2131, 2132, 2133, 2134, 2135, 2136, 2137, 2138, 2139, 2140, 2141, 2142, 2143, 2144, 2145, 2146, 2147, 2148, 2149, 2150, 2151, 2152, 2153, 2154, 2155, 2156, 2157, 2158, 2159, 2160, 2161, 2162, 2163, 2164, 2165, 2166, 2167, 2168, 2169, 2170, 2171, 2172, 2173, 2174, 2175, 2176, 2177, 2178, 2179, 2180, 2181, 2182, 2183, 2184, 2185, 2186, 2187, 2188, 2189, 2190, 2191, 2192, 2193, 2194, 2195, 2196, 2197, 2198, 2199, 2200, 2201, 2202, 2203, 2204, 2205, 2206, 2207, 2208, 2209, 2210, 2211, 2212, 2213, 2214, 2215, 2216, 2217, 2218, 2219, 2220, 2221, 2222, 2223, 2224, 2225, 2226, 2227, 2228, 2229, 2230, 2231, 2232, 2233, 2234, 2235, 2236, 2237, 2238, 2239, 2240, 2241, 2242, 2243, 2244, 2245, 2246, 2247, 2248, 2249, 2250, 2251, 2252, 2253, 2254, 2255, 2256, 2257, 2258, 2259, 2260, 2261, 2262, 2263, 2264, 2265, 2266, 2267, 2268, 2269, 2270, 2271, 2272, 2273, 2274, 2275, 2276, 2277, 2278, 2279, 2280, 2281, 2282, 2283, 2284, 2285, 2286, 2287, 2288, 2289, 2290, 2291, 2292, 2293, 2294, 2295, 2296, 2297, 2298, 2299, 2300, 2301, 2302, 2303, 2304, 2305, 2306, 2307, 2308, 2309, 2310, 2311, 2312, 2313, 2314, 2315, 2316, 2317, 2318, 2319, 2320, 2321, 2322, 2323, 2324, 2325, 2326, 2327, 2328, 2329, 2330, 2331, 2332, 2333, 2334, 2335, 2336, 2337, 2338, 2339, 2340, 2341, 2342, 2343, 2344, 2345, 2346, 2347, 2348, 2349, 2350, 2351, 2352, 2353, 2354, 2355, 2356, 2357, 2358, 2359, 2360, 2361, 2362, 2363, 2364, 2365, 2366, 2367, 2368, 2369, 2370, 2371, 2372, 2373, 2374, 2375, 2376, 2377, 2378, 2379, 2380, 2381, 2382, 2383, 2384, 2385, 2386, 2387, 2388, 2389, 2390, 2391, 2392, 2393, 2394, 2395, 2396, 2397, 2398, 2399, 2400, 2401, 2402, 2403, 2404, 2405, 2406, 2407, 2408, 2409, 2410, 2411, 2412, 2413, 2414, 2415, 2416, 2417, 2418, 2419, 2420, 2421, 2422, 2423, 2424, 2425, 2426, 2427, 2428, 2429, 2430, 2431, 2432, 2433, 2434, 2435, 2436, 2437, 2438, 2439, 2440, 2441, 2442, 2443, 2444, 2445, 2446, 2447, 2448, 2449, 2450, 2451, 2452, 2453, 2454, 2455, 2456, 2457, 2458, 2459, 2460, 2461, 2462, 2463, 2464, 2465, 2466, 2467, 2468, 2469, 2470, 2471, 2472, 2473, 2474, 2475, 2476, 2477, 2478, 2479, 2480, 2481, 2482, 2483, 2484, 2485, 2486, 2487, 2488, 2489, 2490, 2491, 2492, 2493, 2494, 2495, 2496, 2497, 2498, 2499, 2500, 2501, 2502, 2503, 2504, 2505, 2506, 2507, 2508, 2509, 2510, 2511, 2512, 2513, 2514, 2515, 2516, 2517, 2518, 2519, 2520, 2521, 2522, 2523, 2524, 2525, 2526, 2527, 2528, 2529, 2530, 2531, 2532, 2533, 2534, 2535, 2536, 2537, 2538, 2539, 2540, 2541, 2542, 2543, 2544, 2545, 2546, 2547, 2548, 2549, 2550, 2551, 2552, 2553, 2554, 2555, 2556, 2557, 2558, 2559, 2560, 2561, 2562, 2563, 2564, 2565, 2566, 2567, 2568, 2569, 2570, 2571, 2572, 2573, 2574, 2575, 2576, 2577, 2578, 2579, 2580, 2581, 2582, 2583, 2584, 2585, 2586, 2587, 2588, 2589, 2590, 2591, 2592, 2593, 2594, 2595, 2596, 2597, 2598, 2599, 2600, 2601, 2602, 2603, 2604, 2605, 2606, 2607, 2608, 2609, 2610, 2611, 2612, 2613, 2614, 2615, 2616, 2617, 2618, 2619, 2620, 2621, 2622, 2623, 2624, 2625, 2626, 2627, 2628, 2629, 2630, 2631, 2632, 2633, 2634, 2635, 2636, 2637, 2638, 2639, 2640, 2641, 2642, 2643, 2644, 2645, 2646, 2647, 2648, 2649, 2650, 2651, 2652, 2653, 2654, 2655, 2656, 2657, 2658, 2659, 2660, 2661, 2662, 2663, 2664, 2665, 2666, 2667, 2668, 2669, 2670, 2671, 2672, 2673, 2674, 2675, 2676, 2677,
```

```

Thu Jul 17 14:46:11 1997          ACCESS.C          Page 22/26

annel,
    delay(120);
    ret_code = DIOAddGroupAddress(Elements[found_element].ch
                                (BYTE *)&Elements[found_element].e_mac,
                                (BYTE *)&pacau->g_key);
}
else
    /* No group, Delete element */
    del_peb_element(found_element);
}
/* Change PEB database for this element */
if(!ret_code)
{
    replace_peb_element(found_element, new_peb->version);
    MOVb( (unsigned char *)new_peb, (unsigned char *)found, PEB_LEN - 2);
}
return(ret_code);
}

/*****
 * AccessReceive(char *message)
 *
 * Description:
 *      This routine checks to see if we've received any
 *      packets from the MID. If we have, the data is copied
 *      from the ECB to the message and the ECB is returned to t
 *      LSL.
 *
 * Input:
 *      message
 *      r to where to copy data
 *
 * Output:
 *      message and lroinfo filled in if successful
 *
 * Returns:
 *      0      if a packet has been received
 *      *****
 *
 * LONG AccessReceive(char *message)
 *
 * {
 *      ECB *ecb;
 *
 *      /* Extract an ECB from the linked list if one exists */
 *      Disable();
 *      ecb = AccessECBHead;
 *      if (!ecb)
 *      {
 *          /* No ecb. Just return */
 *          Enable();
 *          return(-1);
 *      }
 *      AccessECBHead = ecb->ECB_NextLink;
 *      if (AccessECBHead == 0)
 *          AccessECBTail = 0;
 *
 *      Enable();
 *
 *      /* copy the data past the lroinfo to the message */
 *      MOVb(ecb->ECB_Fragment[0].FragmentAddress, message, ecb->ECB_Fragment[0]
 *      .FragmentLength);
 *
 *      /* return the ecb to the LSL */
 *****/

```

```

# ifdef LOG_ECB_ACTIVITY
#  define LOG_ECB_ACTIVITY (logECBHandle, (logECBHandle, TRUE,
#  define LOG_ECB_ACTIVITY *ACCESS return($081x\n", ecb));
# endif
return(0);
)

/*****
AccessAdd(BYTE *message)
Description:
    This routine is called when a packet is received
    and is responsible for dispatching it.
Input:
    message
    lroInfo
    - packet data
    - lro information
Output:
    Nothing
Returns:
    Nothing
*****/

int AccessAdd(BYTE *message) {
    IndPacket_t *p_packet;
    int packet_type;
    int status;

    p_packet = (IndPacket_t *)message;

    if(p_packet->address[3] == 0x01) /* CAS Individual address */
        switch(p_packet->payload[0]) {
            case SG_PACAU:
                packet_type = PACAU;
                break;
            case SG_ECAU:
                packet_type = ECAU;
                break;
        }
    else if(p_packet->address[0] == 0x03) /* Bypass key */
        switch(p_packet->payload[0]) {
            case PEB_PACKET:
                packet_type = PEB;
                break;
            case GUP_PACKET:
                packet_type = GUP;
                break;
            case PEB_UPDATE_PACKET:
                packet_type = PEB_UPDATE;
                break;
            default:
                packet_type = UNKNOWN;
                break;
        }
    else
        packet_type = UNKNOWN;

    switch(packet_type) {
        case UNKNOWN:
            break;
    }
}

case PACAU:
    status = parse_pacau(p_packet->payload+1, p_packet->length-1);
    break;
case ECAU:
    status = parse_ecau(p_packet->payload+1, p_packet->length-1);
    break;
case PEB:
    status = parse_peb(p_packet->payload+1, p_packet->length-1);
    break;
case GUP:
    status = parse_gup(p_packet->payload+1, p_packet->length-1);
    break;
case PEB_UPDATE:
    status = update_peb(p_packet->payload+1, p_packet->length-1);
    break;
}
return(status);
)

/*****
parse_dacau(BYTE *buf,
int len)
Description:
    Parse the new DACAU from the message received by the mod
    em.
    Replace the old DACAU buffer by the new one if the v
    ersion matches what we have in the CASDBdacau version.
Input:
    buf
    len
    - pointer to the message receive
    - length of the message received
Output:
    Nothing
Returns:
    0 if successful
*****/

static int parse_dacau(BYTE *buf, int len)
{
    unsigned long curr_d_version;
    int ret = 0;

    curr_d_version =
        buf[0] * 256UL +
        buf[1] * 256UL +
        buf[2] * 256UL +
        buf[3] * 256UL +
        256UL;

    if(curr_d_version == CASDBdacau.version) {
        CDBVersion++;
        CASDBdacau.version = curr_d_version;
        CMOV(buf + DACAU_HEAD_LEN, CASDBdacau.p_buffer, len - DACAU_HEAD_LEN);
        CASDBdacau.length = len - DACAU_HEAD_LEN;
        CASDBdacau.entries = CASDBdacau.length / CASDBdacau.entry_len;
        check_df_groups();
    }
    else {
        ret = -1;
    }
}

```



```

return ret;
}

.....

DacauResponse(BYTE *pdata,
               int len,
               timeoutFlag)

Description:
    This is the callback routine pass in to DIOScheduleRecei
    ve()
    which receives the response from the modem. If a timeout
    hadn't occurred, parse the message for the dacau info.

Input:
    pdata          - Pointer to response me
    len            - length of the message
    timeoutFlag    - non-zero if we timed out

Output:
    Nothing

Returns:
    Nothing
.....

static void DacauResponse(BYTE *pdata, int len, int timeoutFlag)
{
    if (timeoutFlag)
    {
        DacauFlag = 1;
        UpdateFileStatus(MSG("Retry: Obtaining Encryption Keys", 258));
        return;
    }
    EnterDebugger();
    if (parse_dacau(pdata, len) == 0)
    {
        WaitingForKeys = FALSE;
        CDBVersion++;
        SaveConfig();
        UpdateFileStatus(MSG("IDLE", 211));
    }
    else
    {
        DacauFlag = 1;
        UpdateFileStatus(MSG("Retry: Obtaining Encryption Keys", 259));
    }
}

.....

GetDacau(void)

Description:
    Initiate the reception of the encryption keys for this a
    dapter
    by constructing a request, sending it to the modem, and
    scheduling a receive routine to receive the response whi
    hopefully contains our encryption key.

Input:
    Nothing
Output:
    Nothing
Returns:
    Nothing
.....

static void GetDacau(void)
{
    int k;
    BYTE c;

    memcpy(&DacauRequest.adapternum, SerialNumPacked, sizeof(ID));
    if (DIOSignText(BYTE *)&DacauRequest, sizeof(DACAUrequest_t) - 8, Dac
    auRequest.sign) != 0)
        return;
    for(k = 0; k < 8; k+=2)
    {
        c = DacauRequest.sign[k];
        DacauRequest.sign[k] = DacauRequest.sign[k+1];
        DacauRequest.sign[k+1] = c;
    }
    SlipSend((BYTE *)&DacauRequest, sizeof(DACAUrequest_t), 1, DIO_GETKEYS_T
    IMEOUT);
    if (DIOScheduleReceive(&DacauResponse, 60, DACAU_RECEIVE) == 0)
        DacauFlag = 0;
}

.....

AccessMain(void *parm)
{
    Description:
        Main access thread which handles conditional access
        packet reception. We'll start out by reading DPC.CFG.
        If the serial number on the adapter is different from th
        at
        in DPC.CFG, we'll call out immediately to get the new ke
        y.
        Then we'll just sleep until there is a packet to
        Input:
            parm          - ignored
        Output:
            Nothing
        Returns:
            Nothing
    }

.....

void AccessMain(void *parm)
{
    LONG sn, ccode;
    ECB ecb = {0, 0, 0, 0, 0, 0, MSG("DRCTPC", 212)};
    ChannelConfig channelCfg;
    BYTE address[8];
    IVTYPE x;
    LONG removedCount;
}

```

```

    parm = parm;

    /*
     * When MUID has been found, Open the conditional access channel.
     */
RegisterWithDriver:
    while(!ExitingFlag)
    {
        AccessAsleep = TRUE;
        delay(500);
        AccessAsleep = FALSE;

        if (DIOBoard != 0)
        {
            removedCount = DIORemovedCount;
            if (AccessChannel != -1)
                DIOOpenChannel(AccessAddress, AccessESR,
                    AccessChannel) == 0)
            {
                DIOGetSN(SerialNum);
                sn = atoi(SerialNum);
                MWSprintf(SerialNum, MSG("%061X", 213), sn);

                pack_mac_addr(SerialNumPacked, 3, SerialNum, 6);
                x = SerialNumPacked(0);
                SerialNumPacked(0) = SerialNumPacked(2);
                SerialNumPacked(2) = x;

                MACbuildAddr(SerialNum, MAC_CAS_IND, 0, (MACAddr
                    if (DIOAddrAddress(AccessChannel, address) == 0)

                channelCfg.CfgChannel = AccessChannel;
                channelCfg.CfgNumAddresses = 1;
                MACbuildAddr(SerialNum, MAC_CAS_IND, 0, (MACAddr

            _t *)channelCfg.CfgAddress);

            ecb.ECB_StackID = MUID_ADD_ADDRESS;
            ecb.ECB_Fragment(0).FragmentAddress = &channelCfg
            if (IoctlMId(FDBBoard, &ecb, FDBControlEntry) =
                = 0)
                break;

        }

    }

    if (ExitingFlag)
    {
        DPCAccessPID = 0;
        return;
    }

    /*
     * Now lets open up the the DPC.CFG file.
     */
ReadConfig():
    while(!ExitingFlag)
    {
        AccessAdd(AccessMessage);
        if (DacauFlag)
            GetDacau();

        DIOCloseChannel(AccessChannel);
        DPCAccessPID = 0;

        AccessAsleep = TRUE;
        SuspendThread(DPCAAccessPID);
        AccessAsleep = FALSE;
        if (removedCount != DIORemovedCount)
            goto RegisterWithDriver;
        continue;
    }
}

```

```

#include "dpcagent.h" /* Our header file */

LONG DIOBoard = 0;
LONG DIORemovedCount = 0;
LONG DIOControlEntry = 0;
struct DriverStatsStructure* DIOStats = 0;

void DIORemove(void)
{
    int i;

    /* Invalidate the DriverIO board. Increment removed count so that
     * other threads can detect that the driver has changed, even if
     * DIOBoard gets filled in. The other threads can then re-register
     * with the new driver.
     */
    DIOStats = 0;
    DIOBoard = 0;
    DIORemovedCount++;

    /* Force the NWSNVT menus to exit so that DPCAGENT can go back
     * to searching for a new adapter before allowing user to choose
     * options.
     */
    for(i = 0; i < 4; i++)
        NWSNGetKey(UCK_ESCAPE_KEY, UCK_NORMAL_KEY, NUTHandle);

    /*
     * Force sleeping threads to wake up so that they can detect that
     * the adapter has been unloaded(DIORemovedCount will be different
     * from their local copy of the last removed count). The threads
     * should then spin(sleep) periodically until a new adapter is
     * present, and then re-register with the adapter if they need to.
     */
    if (AccessAsleep)
        ResumeThread(DPCAccessPID);

    if (InetAsleep)
        ResumeThread(DPCInetPID);

    LONG DIORegisterWithAdapter(char *shortName)
    {
        LONG board;

        for(board = 0; board < NumberOfLANS; board++)
        {
            void (*ControlEntryPoint) (void) = NULL;
            if (CLSIDGetMLIDControlEntry(board,
                &ControlEntryPoint) == SUCCESS)
            {
                struct DriverConfigurationsStructure* config = 0;
                if ((config = (struct DriverConfigurationsStructure *) Co
                    mmAndMId(board, 0, (LONG)ControlEntryPoint)))
                {
                    if (!CStrCmp(config->DShortName, shortName))
                    {
                        ECB ecb = {0, 0, 0, 0, 0, MSG("DRCTPC",
505));

```

```

void (*removeRoutine) () = DIORemove;

ecb.ECB_StackID = MLID_REGISTER_AGENT;
ecb.ECB_Fragment[0].FragmentAddress = &r

```

```

yPoint);

/* Call MLID */
IoctlMId(board, &ecb, (LONG)ControlEntr

```

```

DIOBoard = board;
DIOControlEntry = (LONG)ControlEntryPoi
return(0);

```

```

return(-1);

```

```

void DIODeRegisterAgent(void)
{

```

```

    ECB ecb = {0, 0, 0, 0, 0, MSG("DRCTPC", 506)};
    void (*nullRoutine) () = 0;

```

```

    if (DIOBoard)
    {

```

```

        ecb.ECB_StackID = MLID_REGISTER_AGENT;
        ecb.ECB_Fragment[0].FragmentAddress = &nullRoutine;

```

```

        /* Call MLID */
        IoctlMId(DIOBoard, &ecb, DIOControlEntry);
        DIOBoard = 0;
    }
}

/******
DIOGetsN(char *serialNum)
Description:
This routine Gets the serial number from the adapter.
It is used for explicit requests of packages that are
for sale.
Input:
serialNum - Pointer to where to store seri
al number
Output:
serialNum - filled out if successful
Returns:
0 if successful
*****
LONG DIOGetsN(char *serialNum)
{
    LONG ccode;
    ECB ecb = {0, 0, 0, 0, 0, MSG("DRCTPC", 108)};
    if (!DIOBoard)
        return(-1);

```

```
ecb.ECB_StackID = MLID_GET_SN;  
ecb.ECB_Fragment[0].FragmentAddress = serialNum;
```

```
/* Call MLID */  
ccode = IoctlMId(DIOBoard, &ecb, DIOControlEntry);  
return(ccode);
```

```
DIOsignText(char *textToSign,  
             LONG textLength,  
             char *signature)
```

```
Description:  
This routine uses the adapter to calculate a signature  
for the given text.  
It is used for explicit requests of packages that are  
for sale.
```

```
Input:  
textToSign      - string to be signed  
textLength      - length of string to be signed  
signature        - string to store signature
```

```
Output:  
signature        - filled out if successful
```

```
Returns:  
0 if successful
```

```
LONG DIOsignText(char *textToSign,  
                 LONG textLength,  
                 char *signature)
```

```
LONG ccode;  
LONG fragment[3];  
ECB ecb = {0, 0, 0, 0, 0, MSG("DRCTPC", 139)};
```

```
if (!DIOBoard)  
    return(-1);
```

```
fragment[0] = (LONG)textToSign;  
fragment[1] = textLength;  
fragment[2] = (LONG)signature;
```

```
ecb.ECB_StackID = MLID_SIGN_TEXT;  
ecb.ECB_Fragment[0].FragmentAddress = fragment;
```

```
/* Call MLID */  
ccode = IoctlMId(DIOBoard, &ecb, DIOControlEntry);  
return(ccode);
```

```
EXPORTED FUNCTION
```

```
DIOGetMIdStats(struct DriverStatsStructure *stats)
```

```
Description:  
This routine fills in a pointer to the MLID stats  
table.
```

```
Input:  
stats           - Pointer to where to store poin  
ter to stats table
```

```
Output:  
stats filled in if successful
```

```
Returns:  
0 if MLID is active
```

```
int DIOGetMIdStats(struct DriverStatsStructure *stats)
```

```
if (!DIOBoard)  
    return(-1);
```

```
*stats = DIOStats = (struct DriverStatsStructure *)  
CommandMId(DIOBoard, 1, DIOControlEntry);
```

```
return(0);
```

```
int DIOGetMIdConfig(struct DriverConfigurationsStructure *config)
```

```
if (!DIOBoard)  
    return(-1);
```

```
*config = (struct DriverConfigurationsStructure *)  
CommandMId(DIOBoard, 0, DIOControlEntry);
```

```
return(0);
```

```
DIOOpenChannel(BYTE *address, int (*est)(), LONG *channel)
```

```
Description:
```

```
This routine attempts to open an adapter channel  
the passed in bypass address, such as 0f 00 00 00.
```

```
Input:
```

```
address         - Bypass  
est             - Bypass
```

```
- ESR address for this channel  
channel         - where
```

```
channel number is returned
```

```
Output:  
channel is filled out if successful
```

```
Returns:  
0 if channel was opened
```

```
LONG DIOOpenChannel(BYTE *address, int (*est)(), LONG *channel)
```

```
ChannelConfig cfg;
```

```

LONG
ECB ecb = (0, 0, 0, 0, 0, 0, MSG("DRCTPC", 142));

if (!DIOBoard) {
    if (DIORegisterWithAdapter(DPCName))
        return(-1);
}

/* Initialize channel to 0. MLID will overwrite this */
cfg.CfgChannel = 0;

/* Point ESR to our packet handler */
cfg.CfgESR = esr;

/* Number of addresses to add */
cfg.CfgNumAddresses = 1;

/* Address of 0f 00 00 00 00 00 */
CMovB(address, cfg.CfgAddress, 6);

ecb.ECB_StackID = MLID_OPEN_CHANNEL;
ecb.ECB_Fragment[0].FragmentAddress = &cfg;

/* Call MLID */
ccode = IoctlMlId(DIOBoard, &ecb, DIOControlEntry);
if (ccode == 0)
{
    /* Store channel for close channel, add addr and delete addr */
    *channel = cfg.CfgChannel;
}
return(ccode);

}

/*****
*
* DIOCloseChannel(LONG channel)
*
* Description:      This routine closes a previously opened channel.
*
* Input:           channel
*
* to close
*
* Output:          nothing
*
* Returns:         0          if channel was closed
*
* *****/
LONG DIOCloseChannel(LONG channel)
{
    LONG ccode;
    ECB ecb = (0, 0, 0, 0, 0, 0, MSG("DRCTPC", 157));

    if (!DIOBoard)
        return(-1);

    ecb.ECB_StackID = MLID_CLOSE_CHANNEL;
    ecb.ECB_Fragment[0].FragmentAddress = &channel;
    ccode = IoctlMlId(DIOBoard, &ecb, DIOControlEntry);

    return(ccode);
}

/*****
*
* DIODeletAddress(BYTE *address)
*
* Description:      This routine deletes the filter address from the MLID.
*
* Input:           address
*
* etc
*
* Output:          Nothing
*
* Returns:         0 if successful
*
* *****/
LONG DIODeletAddress(LONG channel, BYTE *address)
{
    ChannelConfig channelCfg;
    LONG ccode;
    ECB ecb = (0, 0, 0, 0, 0, 0, MSG("DRCTPC", 180));

    if (!DIOBoard)
        return(-1);

    channelCfg.CfgChannel = channel;
    channelCfg.CfgNumAddresses = 1;
    CMovB(address, channelCfg.CfgAddress, 6);

    ecb.ECB_StackID = MLID_DEL_ADDRESS;
    ecb.ECB_Fragment[0].FragmentAddress = &channelCfg;
    ccode = IoctlMlId(DIOBoard, &ecb, DIOControlEntry);

    return(ccode);
}

LONG DIODelGroupAddress(LONG channel, BYTE *address, BYTE *groupAddress)
{
    ChannelConfig channelCfg;
    LONG ccode;
    ECB ecb = (0, 0, 0, 0, 0, 0, MSG("DRCTPC", 209));
    BYTE key[8];

    channelCfg.CfgChannel = channel;
    channelCfg.CfgNumAddresses = 1;
    CMovB(address, channelCfg.CfgAddress, 8);

    ecb.ECB_StackID = MLID_ADD_ADDRESS;
    ecb.ECB_Fragment[0].FragmentAddress = &channelCfg;
    ccode = IoctlMlId(DIOBoard, &ecb, DIOControlEntry);

    return(ccode);
}

```

```

if (!DIOBoard)
    return(-1);

channelCfg.CfgChannel = channel;
channelCfg.CfgNumAddresses = 1;
CMOVb(address, channelCfg.CfgAddress, 8);
CMOVb(groupAddress, key, 8);
reverse_key((BYTE*)(key));
CMOVb(key, channelCfg.CfgGroupKey, 8);
CMOVb(magicKey, channelCfg.CfgElementKey, 8);

ecb.ECB_StackID = MLID_ADD_ADDRESS;
ecb.ECB_Fragment[0].FragmentAddress = &channelCfg;
ecode = IoctlMId(DIOBoard, &ecb, DIOControlEntry);
return(ecode);
}

int DIOAddHAddr(unsigned char channel, BYTE *hAddr)
{
    chunk key;
    int h1_id;
    ret = CAS_ERROR;
    ChannelConfig channelCfg;
    ECB ecb = {0, 0, 0, 0, 0, MSG("DRCTPC", 471)};

    if (!DIOBoard)
        return(-1);

    for(i = 0; i < 3; i++)
        h1_id[i] = 0x00;
        make_h1_key(&key);
    channelCfg.CfgChannel = channel;
    channelCfg.CfgNumAddresses = 1;

    CMOVb(h1Addr, channelCfg.CfgAddress, 8);
    /* Some strange things ... */
    reverse_key((BYTE*)(key));
    CMOVb(&key, channelCfg.CfgGroupKey, 8);
    CMOVb(&key, channelCfg.CfgElementKey, 8);

    channelCfg.CfgChannel = FDBChannel;
    channelCfg.CfgESR = FDB_ESR;
    channelCfg.CfgNumAddresses = 1;
    CMOVb(&addr, channelCfg.CfgAddress, 8);
    CMOVb(&pacu->g_key, key, 8);
    reverse_key(&key);
    CMOVb(key, channelCfg.CfgGroupKey, 8);
    CMOVb(magicKey, channelCfg.CfgElementKey, 8);

    channelCfg.CfgESR = (int (*)( ))0xffffffff;
    ecb.ECB_StackID = MLID_ADD_ADDRESS;
    ecb.ECB_Fragment[0].FragmentAddress = &channelCfg;
    CMOVb(&addr, node->file_address, 6);

    ret = IoctlMId(DIOBoard, &ecb, DIOControlEntry);

    if ((ret == WBICddAddAddress
        (BICDD_CHANNEL_CONFIG_FAR *)(&channel_config)) != CAS_OK)
        return ret;
}

int DIORegisterSend(int (*sendRoutine)(TCB *))
{
    int DIOObtainReturnTCB(void (**returnTCBRoutine)(TCB *))
    {
        ECB ecb = {0, 0, 0, 0, 0, MSG("DRCTPC", 147)};

        if (!DIOControlEntry)
            return(-1);

        ecb.ECB_StackID = MLID_RETURN_TCB_ROUTINE;
        ecb.ECB_Fragment[0].FragmentAddress = returnTCBRoutine;

        /* Call MLID */
        IoctlMId(DIOBoard, &ecb, DIOControlEntry);
        return(0);
    }

    int DPGGetIPAddress(LONG *ip) {
        LONG id;
        LONG (*ctl)(LONG board, ...);
        char buf[80];
        char *s = buf;

        if (CLISGetStackIDFromName("002IP", &id))
            return 1;
        if (CLISGetProtocolControlEntry(id, DIOBoard, &ctl))
            return 2;
        if (GetProtocolStringForBoard(ctl, DIOBoard, buf))
            return 3;
        s = strcpy(buf, "0123456789");
        if (!s)
            return 4;
        *ip = inet_addr(s);
        if (*ip == (LONG)(-1))
            return 5;
        return 0;
    }
}

```

```
#include "dpcagent.h"
#include <string.h>
#include <stdlib.h>
```

```
LONG DPCMaxConnections = 3;
int PackageDelivery = FALSE;
```

```
DlCfg_t DlCfg = {
```

```
1330, // freq
(198 << 24) | (77 << 16) | (117 << 8) | 21, // ip_address
(198 << 24) | (77 << 16) | (117 << 8) | 66, // gateway_address
1500, // mtu
MSG("ATDRI-800-333-8071", 100), // tnet_phone_num
MSG("ATDRI-800-825-3954", 187), // pdeliv_phone_num
300, // dialout_prefix
2, // tnet_inactivity_timer
1, // pdeliv_inactivity_timer
120, // modem_type
ca in buffer // packet_lifetime - Max time to keep da
```

```
60, // call_setup_timeout
MSG("ATH0", 189), // hangup_str
MSG("NO CARRIER", 190), // disconnect_str
MSG("+++", 191), // escape_str
MSG("CONNECT", 192), // connect_str
MSG("ATE1Q0V1X4&C1&D2 S7=60 S11=55", 193), // init_str
1024, // max_db_entries
5, // tnet_baud_index(19200)
0, // pdeliv_baud_index(2400)
1024 * 8, // async_buffer_size(8K)
FALSE, // auto_login
OUT_SLIP, // wait_for_1..wait_for_9
TRUE, // send_1..send_9
10, 5, 5, 5, 5, 5, 5, 5, // out_protocol
1400, // obsolete(was tunnel)
0, // wait_timeout_1..wait_timeout_9
"00000000", // ppp_login
"00000000", // ppp_passwd
"00000000", // ppp_mru
"00000000", // ppp_accm
"00000000", // base_license
"00000000", // key
"00000000", // net_interface
"00000000", // net_addr
"00000000", // add_license
```

```
/*.....*/
EXPORTED FUNCTION
```

```
DPCUpdateConfig(void)
```

```
Description:
```

```
This routine opens \DIRECTPC\DB\MODEM.CFG and updates our global
structures. If the file doesn't exist, or if its an older versio
n
(because its smaller), read in what you can and write out a new
one.
```

```
Input:
```

```
Output:
```

```
Returns:
0 if successful
-1 unable to open or create file
-2 unable to update file
```

```
void ConfigSanityCheck(int handle)
```

```
{
    int changeFlag = 0;
```

```
    int i;
    LONG *timeout;
```

```
    if (DlCfg.wait_timeout_1 < 2 || DlCfg.wait_timeout_1 > 60)
```

```
    {
        changeFlag++;
        DlCfg.wait_timeout_1 = 10;
```

```
    }
    for (i = 0, timeout = &DlCfg.wait_timeout_2; i < 8; i++, timeout++)
```

```
    {
        if (*timeout < 2 || *timeout > 60)
```

```
    {
        changeFlag++;
        *timeout = 5;
```

```
    }
    if (changeFlag)
```

```
    {
        lseek(handle, 0, SEEK_SET);
        write(handle, &DlCfg, sizeof(DlCfg));
```

```
    }
    int DPCUpdateConfig(void)
```

```
{
    int ccode = -1;
    int handle;
```

```
    handle = open(MSG("SYS:DIRECTPC\\DB\\MODEM.CFG", 194),
        O_RDWR | O_CREAT,
        S_IRWRITE | S_IRREAD);
```

```
    if (handle != -1)
```

```
    {
        if (read(handle, &DlCfg, sizeof(DlCfg)) != sizeof(DlCfg))
```

```
    {
        lseek(handle, 0, SEEK_SET);
        if (write(handle, &DlCfg, sizeof(DlCfg)) != sizeof(DlCfg))
```

```
    {
        ccode = -2;
```

```
    }
    ConfigSanityCheck(handle);
    close(handle);
```

```
    ccode = 0;
```

```
    DlCfg.ip_address = hton(DlCfg.ip_address);
    DlCfg.gateway_address = hton(DlCfg.gateway_address);
```

```
    return(ccode);
```

```
void DPCUpdateConfigFile(void) {
```

```

int handle;

DioCfg.ip_address = htonl(DioCfg.ip_address);
DioCfg.gateway_address = htonl(DioCfg.gateway_address);
handle = open(MSG(SYS:DIRECTPC\DB\MODEM.CFG", 335),
              O_RDWR | O_CREAT,
              S_IRWRITE | S_IRREAD);
if (handle != -1)
{
    write(handle, &DioCfg, sizeof(DioCfg));
    close(handle);
}
DioCfg.ip_address = htonl(DioCfg.ip_address);
DioCfg.gateway_address = htonl(DioCfg.gateway_address);

#define SerialWarn(s) sprintf(warnbuf, "\r\nDPCN: detected a bad serial number: %0.8s\r\n", (char*)(s)), ConsolePrintf(warnbuf), RingTheBell()

static inline unsigned long find_and_clear_low_bit(unsigned long* val) {
    if (low == 0)
        return 0;
    *val &= low - 1;
    return (*val ^ low);
}

static inline int parity(LONG serial) {
    int i;
    for (i = 0; find_and_clear_low_bit(&serial); ++i)
        return (i & 1);
}

static inline int UserCount(BYTE* s) {
    LONG serial = 0;
    int shift;

    s += 3;
    for (shift = 16; shift >= 0; shift -= 4) {
        serial |= ((*s - (('s' >= 'A') ? 56 : 0x30)) << shift;
        ++s;
    }
    return 5 * (((serial & 0x00002) >> 1) |
                ((serial & 0x20000) >> 16) |
                ((serial & 0x2000) >> 11) |
                ((serial & 0x00040) >> 3));
}

void DPCSetMaxConnections(LONG* sum) {
    char warnbuf[120];
    int users;
    int pd = 0;
    int i;

    sum[0] = sum[1] = (-1);

    /* re-read modem.cfg file */
    i = DioCfg.ip_address;
    DPCupdateConfig();
    DioCfg.ip_address = i;

    if (strcmp(DioCfg.base_license, "Helius, Inc.", 8) == ESUCCESS) {
        DPCMaxConnections = 108;
        PackageDelivery = 1;
        memcpy(sum, DioCfg.base_license, 2 * sizeof(LONG));
        return;
    }

    /* check for old license info */
    if (atoi(DioCfg.base_license) < 02) {
        sprintf(warnbuf,
            "\r\nDPCN: deactivated old serial number: %0.8s\r\n",
            DioCfg.base_license);
        ConsolePrintf(warnbuf);
        RingTheBell();
        return;
    }

    /* handle base license first */
    memcpy(sum, DioCfg.base_license, 2 * sizeof(LONG));
    if (parity(sum[0]) == 0) {
        SerialWarn(DioCfg.base_license);
        return;
    }
    if (parity(sum[1]) == 0) {
        SerialWarn(DioCfg.base_license);
        return;
    }
    users = UserCount(DioCfg.base_license);
    if (DioCfg.base_license[2] == '*')
        pd = 1;

    /* now handle additive licenses */
    for (i = 0; i < 9; ++i) {
        int j;
        LONG serial[2];
        if (DioCfg.add_license[i][0] == 0)
            continue;
        /* check for duplicate license */
        for (j = 1 - 1; j >= 0; --j) {
            if (memcmp(DioCfg.add_license[i],
                      DioCfg.add_license[j],
                      sizeof(DioCfg.add_license[i])) == ESUCCESS) {
                memset(DioCfg.add_license[i], 0, sizeof(DioCfg.add_license[i]));
                DPCupdateConfigFile();
                sprintf(warnbuf,
                    "\r\nDPCN: deleted duplicate license number: %0.8s\r\n",
                    DioCfg.add_license[j]);
                ConsolePrintf(warnbuf);
                RingTheBell();
                goto nextlicense;
            }
        }
        memcpy(serial, DioCfg.add_license[i], sizeof(serial));
        if (parity(serial[0]) == 1) {
            SerialWarn(DioCfg.add_license[i]);
            return;
        }
        if (parity(serial[1]) == 1) {
            SerialWarn(DioCfg.add_license[i]);
            return;
        }
        users += UserCount(DioCfg.add_license[i]);
        sum[0] += serial[0];
        sum[1] += serial[1];
        if (DioCfg.add_license[i][2] == '*')

```


Thu Jul 17 14:46:12 1997

license.c

Page 5

pd = 1;
nextLicense;

DPOMaxConnections = users * 4;
PackageDelivery = pd;

```

#include <dpccagent.h> /* Our header file */

#define USE_AIO_DEADMAN 0
#define TRACE_STATE 0

static char * printables = MSG("A-Za-z0-9-._!@#$%^&*()+=[]{}|:;'\",.<>?\\/\\"
523);
#define dial_chars printables
#define modem_control_chars printables

int AIOportHandle = -1;
LONG AIOwriteBufferSize = 0;
int AIOGlobalPort = 0;
static BYTE AIObaudrateDefinitions[] =
(
    AIO_BAUD_2400, /* 0 */
    AIO_BAUD_3600, /* 1 */
    AIO_BAUD_4800, /* 2 */
    AIO_BAUD_7200, /* 3 */
    AIO_BAUD_9600, /* 4 */
    AIO_BAUD_19200, /* 5 */
    AIO_BAUD_38400, /* 6 */
    AIO_BAUD_57600, /* 7 */
    AIO_BAUD_115200 /* 8 */
);

int DIOstate = DIOS_IDLE;
LONG DIOTimer = 0;
LONG DIOPacketLifeTimer = 0;

LONG DIOConn = DIO_CONN_IDLE;
LONG DIONextConn = DIO_CONN_IDLE;

LONG DIOPXmtCount = 0;
LONG DIOPMaxBufferSize = DLOBUF_SIZE;
LONG DIOPXmtBuffer[DLOBUF_SIZE];
LONG DIOPInactivityTimer = 10 * 19;

LONG DIOXmtCount = 0;
LONG DIOIxmBufferSize = DLOBUF_SIZE;
LONG DIOIxmBuffer[DLOBUF_SIZE];
LONG DIOInactivityTimer = 60 * 19;

LONG DIOLastKnownTickCount;
LONG DIOLrcvCount = 0;
LONG DIOLrcvIndex = 0;
LONG DIOLreadIndex = 0;
LONG DIOLrcvBuffer[DLOBUF_SIZE];
LONG DIOLCommandIndex = 0;
LONG DIOLCommandBuffer[DLOCMBUF_SIZE];
LONG DIOLastDCD = 0;

char ConnectBaudStr[DLOCMBUF_SIZE] = (0);

static char *DIOCompareStr(DLOCMBUF_SIZE) = (0);

/*
 * DIOcfg.connect_str,
 * DIOcfg.disconnect_str,
 */
MSG("OK", 219),
MSG("BUSY", 220),
MSG("NO DIAL TONE", 221),
MSG("RING", 222),

);

/*
 * Variables used by DIOScheduleReceive().
 */
void (*DIOCallBack)(BYTE *pdata, int len, int timeoutFlag) = 0;
int DIOCallBackTimeout = 0;
int DIOCallBackWait = 0;
BYTE DIOCallBackBuffer[4000];
int DIOCallBackIndex = 0;
int DIOCallBackStarted = 0;
int DIOCallBackEscape = 0;
int DIOCallBackType = 0;

/*
 * RS232.NLM Default Init String
 */
//BYTE
ModemInitString[] = MSG("ATH0Q0V1X4S0",

/*
 * Digital DS144FVM, Multitech Auto Reliable and Practical
 * Peripherals V.34 Default Init String
 */
//BYTE
ModemInitString[] = "ATE1Q0V1X4dC1d2&K0 S7=60 S

/*
 * Hayes-Compatible Modem Default Init String
 */
//BYTE
ModemInitString[] = MSG("ATE1Q0V1X4dC1d2 S7=60

/*
 * Intel 144e Faxmodem and Motorola UDS V.3225
 * Default Init String
 */
//BYTE
ModemInitString[] = "ATE1Q0V1X4dC1d2\G S7=60 S1

/*
 * U.S. Robotics Default Init String
 */
//BYTE
ModemInitString[] = MSG("ATE1Q0V1X4dC1d2&K0&H0&

/*
 * Internal function prototypes */
void initializeIO(void);
void SendData(BYTE *data, LONG length);
static void DIOSartTimer(LONG ticks);
//void DIOSstopTimer(void);
static void DIOSropPacketLifeTimer(void);
static void DIOSartPacketLifeTimer(void);
static void StateMachine(int event);
static void WriteCommXmtBuffer(void);

void HexAsciiDump(const unsigned char *buffer, unsigned int len)
{
    char display[80];
    while (len >= 16)
    {
        Nprintf(MSG("%02x %02x %02x %02x %02x %02x %02x %02x %02x %02x %02x %02x %02x %02x %02x %02x",
        x %02x %02x %02x %02x %02x %02x %02x %02x %02x %02x %02x %02x %02x %02x %02x %02x", 127),
        buffer[0],
        buffer[1],

```

```

buffer[2],
buffer[3],
buffer[4],
buffer[5],
buffer[6],
buffer[7],
buffer[8],
buffer[9],
buffer[10],
buffer[11],
buffer[12],
buffer[13],
buffer[14],
buffer[15],
isprint(buffer[0]) ? buffer[0] : ' ',
isprint(buffer[1]) ? buffer[1] : ' ',
isprint(buffer[2]) ? buffer[2] : ' ',
isprint(buffer[3]) ? buffer[3] : ' ',
isprint(buffer[4]) ? buffer[4] : ' ',
isprint(buffer[5]) ? buffer[5] : ' ',
isprint(buffer[6]) ? buffer[6] : ' ',
isprint(buffer[7]) ? buffer[7] : ' ',
isprint(buffer[8]) ? buffer[8] : ' ',
isprint(buffer[9]) ? buffer[9] : ' ',
isprint(buffer[10]) ? buffer[10] : ' ',
isprint(buffer[11]) ? buffer[11] : ' ',
isprint(buffer[12]) ? buffer[12] : ' ',
isprint(buffer[13]) ? buffer[13] : ' ',
isprint(buffer[14]) ? buffer[14] : ' ',
isprint(buffer[15]) ? buffer[15] : ' ',
buffer += 16;
len -= 16;
}
if (len)
/* the basic theory here is to build the buffer in place and the
unsigned int n = 0;
register char* d = display;
while (n < len)
{
    nmsprintf(d, MSG("%02x ", 483), buffer[n]);
    d += 3;
    display[(16 * 3 + 2) + n] = isprint(buffer[n]) ? buffer[
++n;
}
display[(16 * 3 + 2) + len] = 0;
memset(d, ' ', (16 - len) * 3 + 2);
d = display + (16 / 2 * 3);
memmove(d + 2, d, sizeof(display) - (16 / 2 * 3) - 2);
d[0] = d[1] = ' ';
d = display + (16 * 3) + 4 + 8;
memmove(d + 2, d, 9);
d[0] = d[1] = ' ';
puts(display);
}
}
void DIOUpdateModemStr( void )
{
    if (DIOState == DIOS_IDLE)
        UpdateModemStr(MSG("Modem Status: IDLE
\n", 201));
}

```

```

else if (DIOState == DIOS_INIT)
    UpdateModemStr(MSG("Modem Status: Initializing Modem
\n", 240));
else if (DIOState == DIOS_DIAL)
    UpdateModemStr(MSG("Modem Status: Dialing
\n", 236));
else if (DIOState == DIOS_REDL)
    UpdateModemStr(MSG("Modem Status: Redialing
\n", 235));
else if (DIOState == DIOS_CONN)
{
    if (DIOConn == DLO_CONN_PACKAGE)
        UpdateModemStr(MSG("Modem Status: Connected to Package D
\n", 241));
    else
    {
        if (ConnectBaudStr[0])
            BYTE connectStr[80];
            nmsprintf(connectStr, MSG("Modem Status: Connect
ed to Internet at%.24s\n", 473), ConnectBaudStr);
            UpdateModemStr(connectStr);
        }
    }
    else
    {
        if (DIOConn == DLO_CONN_PACKAGE)
            UpdateModemStr(MSG("Modem Status: Disconnecting from Pac
kage Delivery
\n", 101));
        else
            UpdateModemStr(MSG("Modem Status: Disconnecting from Int
ernet
\n", 475));
    }
}
int DIOGetWriteBufferSize( void )
{
    LONG writeCount = 0;
    if (!AIOWriteBufferSize)
        return(2048);
    AIOGetPortStatus(AIOPortHandle, &writeCount, NULL, NULL, NULL, NUL
L);
    return(AIOWriteBufferSize - writeCount);
}
int DIOAndConnEmpty( void )
{
    if (DIOConn == DLO_CONN_PACKAGE)
        return(DIOpXmitCount == 0);
    else
        return(DIOiXmitCount == 0);
}
void DIOFlushReceive(void)
{
    AIOFlushBuffers(AIOPortHandle, AIO_FLUSH_READ_BUFFER);
}

```

```

    DloRecvCount = 0;
    DloRecvIndex = 0;
    DloReadIndex = 0;
}

void DloStartConn(int timeout)
{
    if (DloNextConn == DLO_CONN_IDLE)
    {
        /* Assume package delivery connection first */
        DloNextConn = DLO_CONN_PACKAGE;

        if (timeout == DLO_PACKAGE_TIMEOUT)
        {
            DloPinactivityTimer = DloCfg.pdeliv_inactivity_timer * 1
9;
        }
        else if (timeout == DLO_INET_TIMEOUT)
        {
            DloPinactivityTimer = DloCfg.tinet_inactivity_timer * 19
;
            DloNextConn = DLO_CONN_INET;
        }
        else if (timeout == DLO_GETKEYS_TIMEOUT)
        {
            DloPinactivityTimer = 30 * 19;
        }
        else if (timeout > 2)
        {
            DloPinactivityTimer = timeout * 19;
        }
        else
        {
            DloPinactivityTimer = 2 * 19;
        }
    }

    if (DloState == DLOS_CONN && DloNextConn == DloConn)
    {
        DloNextConn = DLO_CONN_IDLE;
        StateMachine(DLOE_SEND);
        return;
    }

    if (DloConn == DLO_CONN_IDLE)
    {
        StateMachine (DLOE_SEND);
        DloConn = DloNextConn;
        DloNextConn = DLO_CONN_IDLE;
    }
    else if (DloConn == DLO_CONN_INET)
    {
        /* Package waiting for internet. Cause it to timeout quickly */
        DloStartTimer(19);
    }
}

int BaudRateHandler(int option, void *parameter)
{
    parameter = parameter;
    return option;
}

static void NoSortHandler(LIST *head, LIST *tail, NUTInfo *handle)
{
    head = head;
    tail = tail;
    handle = handle;

    return;
}

void PDConfigure(void)
{
    int i;
    void (*oldSortFunction)(LIST *, LIST *, NUTInfo *);
    int save;
    DloCfg_t tmpDloCfg;
    MFCNTROL *mfcctl;
    int baud;

    CMovB(&DloCfg, &tmpDloCfg, sizeof(DloCfg_t));

    NWSInitForm(NUTHandle);

    NWSGetListSortFunction(NUTHandle, &oldSortFunction);
    NWSSetListSortFunction(NUTHandle, NoSortHandler);

    i = 0;
    NWSAppendCommentField(i, 2, MSG("Package Phone : ", 311), NUTH
andle);
    NWSAppendStringField(i, 28, 30, NORMAL_FIELD, tmpDloCfg.pdeliv_phone_num
    dial_chars, F_NO_HELP, NUTHandle);

    i++;
    baud = tmpDloCfg.pdeliv_baud_index;
    NWSAppendCommentField(i, 2, MSG("Package Baud : ", 313), NUTH
andle);
    mfcctl = NWSInitMenuField(InxMSG("Baud Rate", 314), 10, 40, BaudRateHand
ler, NUTHandle);
    NWSAppendTokenField(mfcctl, InxMSG("2400", 315), 0, NUTHandle);
    NWSAppendTokenField(mfcctl, InxMSG("3600", 316), 1, NUTHandle);
    NWSAppendTokenField(mfcctl, InxMSG("4800", 317), 2, NUTHandle);
    NWSAppendTokenField(mfcctl, InxMSG("7200", 318), 3, NUTHandle);
    NWSAppendTokenField(mfcctl, InxMSG("9600", 319), 4, NUTHandle);
    NWSAppendTokenField(mfcctl, InxMSG("19200", 320), 5, NUTHandle);
    NWSAppendTokenField(mfcctl, InxMSG("38400", 321), 6, NUTHandle);
    NWSAppendTokenField(mfcctl, InxMSG("57600", 322), 7, NUTHandle);
    NWSAppendTokenField(mfcctl, InxMSG("115200", 323), 8, NUTHandle);
    NWSAppendMenuField(i, 28, NORMAL_FIELD, &baud, mfcctl, NULL, NUTH
    i++;
    NWSAppendCommentField(i, 2, MSG("Package Inactivity(sec) : ", 324), NUTH
andle);
    NWSAppendIntegerField(i, 28, NORMAL_FIELD, (int *)&tmpDloCfg.pdeliv_inac
tivity_timer, 1, 65535, F_NO_HELP, NUTHandle);

    i++;
    NWSAppendCommentField(i, 2, MSG("Max Database Entries : ", 327), NUTH
andle);
    NWSAppendIntegerField(i, 28, NORMAL_FIELD, (int *)&tmpDloCfg.max_db_entr
ies, 1, 65535, F_NO_HELP, NUTHandle);

    i++;
    save = NWSeditPortalForm(InxMSG("Package Delivery Configuration Editor",
    308),
    12, 40,
    /* center line, column */
    i, 76,

```

```

/* form height, width */
F_NO_VERIFY, F_NO_HELP, /* Control flags, help message */
InMSG("Save Changes?", 328),
NUTHandle); /* Confirm message, handle */

```

```

NMSSetSortFunction(NUTHandle, oldSortFunction);
NMSDestroyForm(NUTHandle);

```

```

if (!save)
    return;

```

```

tmpDlCfg.pdev_iv_baud_index = baud;
CMoveB(&tmpDlCfg, &DlCfg, sizeof(DlCfg_t));

```

```

DPCUpdateConfigFile();

```

```

LONG ModifyPPPConfig(FIELD *fp, int key, int *changed, NUTInfo *handle)
{

```

```

    int i;
    DlCfg_t *tmpDlCfg;
    LONG save;

```

```

    key = key;
    changed = changed;
    handle = handle;

```

```

    tmpDlCfg = (DlCfg_t *)fp->customData;

```

```

    if (NMSPushList(NUTHandle) == 0)
        return K_SELECT;

```

```

    NMSInitForm(NUTHandle);

```

```

    i = 0;

```

```

    NMSAppendCommentField(i, 2, MSG("Authentication User Name: ", 516), NUTHandle);
    NMSAppendStringField(i, 28, 30, NORMAL_FIELD, tmpDlCfg->ppp_login,
        printables, F_NO_HELP, NUTHandle);

```

```

    i++;

```

```

    NMSAppendCommentField(i, 2, MSG("Authentication Password: ", 578), NUTHandle);
    NMSAppendStringField(i, 28, 30, NORMAL_FIELD, tmpDlCfg->ppp_password,
        printables, F_NO_HELP, NUTHandle);

```

```

    i+=2;

```

```

    NMSAppendCommentField(i, 2, MSG("Maximum Receive Unit : ", 579), NUTHandle);
    NMSAppendIntegerField(i, 28, NORMAL_FIELD, (int *)&tmpDlCfg->ppp_mru, 64, 16384, F_NO_HELP, NUTHandle);

```

```

    i++;
    NMSAppendCommentField(i, 2, MSG("Asynch. Control Char Map: 0x", 580), NUTHandle);
    NMSAppendHexField(i, 30, NORMAL_FIELD, (int *)&tmpDlCfg->ppp_accm, 0, 0xfffffff, F_NO_HELP, NUTHandle);

```

```

    i++;
    save = NMSEditPortalForm(InMSG("PPP Configuration Editor", 510),
        12, 40, /* center line, column */
        1, 78,

```

```

/* form height, width */
F_NO_VERIFY, F_NO_HELP, /* Control flags, help message */
InMSG("Save Changes?", 511),
NUTHandle); /* Confirm message, handle */

```

```

if (!save)
{
    CMoveB(tmpDlCfg->ppp_login, DlCfg.ppp_login, (30*2)+4+4);
    DPCUpdateConfigFile();
}

```

```

NMSDestroyForm(NUTHandle);
NMSPopList(NUTHandle);
return K_SELECT;

```

```

LONG ModifyNetConfig(FIELD *fp, int key, int *changed, NUTInfo *handle)
{

```

```

    int i;
    DlCfg_t *tmpDlCfg = (DlCfg_t *)fp->customData;
    LONG interface;
    char hw_addr[18];
    int save;

```

```

    key = key;
    changed = changed;
    handle = handle;
    /* not used */
    /* not used */
    /* not used */

```

```

    if (NMSPushList(NUTHandle) == 0)
        return K_SELECT;

```

```

    NMSInitForm(NUTHandle);

```

```

    retry:

```

```

    i = 0;
    interface = tmpDlCfg->net_interface;
    NMSAppendCommentField(i, 2, "Interface: ", NUTHandle);
    NMSAppendUnsignedIntegerField(i, 35, NORMAL_FIELD,
        &interface,
        0, 256,
        F_NO_HELP, NUTHandle);

```

```

    i++;

```

```

    NMSAppendCommentField(i, 2, "Router Mac Address: ", NUTHandle);
    sprintf(hw_addr, "%02x-%02x-%02x-%02x-%02x-%02x",
        tmpDlCfg->net_addr[0],
        tmpDlCfg->net_addr[1],
        tmpDlCfg->net_addr[2],
        tmpDlCfg->net_addr[3],
        tmpDlCfg->net_addr[4],
        tmpDlCfg->net_addr[5]);
    NMSAppendStringField(i, 35, 17, NORMAL_FIELD,
        hw_addr,
        "0.9A.Fa.f-",
        F_NO_HELP, NUTHandle);

```

```

    i++;

```

```

    save = NMSEditPortalForm(InMSG("Network Route Configuration Editor", 679),

```

```

        12, 40, /* center line & column */
        1, 78, /* form height & width */
        F_NO_VERIFY, F_NO_HELP,
        NUTHandle);
    if (!save)

```

```

LONG net_addr[6];
void (*ControlEntryPoint)(void) = 0;
struct DriverConfigurationsStructure* dvrcfg = 0;
if (sscanf(hw_addr, "%x-%x-%x-%x-%x-%x",
    &net_addr[0],
    &net_addr[1],
    &net_addr[2],
    &net_addr[3],
    &net_addr[4],
    &net_addr[5]) != 6)
{
    NMSAlert(12, 40, NUTHandle,
        InxMSG("Format error in Mac Address", 680));
    goto retry;
}
for (i = 0; i < 6; ++i)
    tmpDlCfg->net_addr[i] = (BYTE)net_addr[i];
if (CUSTGetMLIDControlEntry(interface,
    &controlEntryPoint))
{
    NMSAlert(12, 40, NUTHandle,
        InxMSG("Interface not found", 681));
    goto retry;
}
dvrcfg = (struct DriverConfigurationsStructure *)
    CommandMId(interface, 0, (LONG)ControlEntryPoint);
if (!dvrcfg)
{
    NMSAlert(12, 40, NUTHandle,
        InxMSG("Could not retrieve Interface Configurati
ion Table", 682));
    goto retry;
}
if ((dvrcfg->DModeFlags & (1 << 6)) == 0)
{
    NMSAlert(12, 40, NUTHandle,
        InxMSG("Interface does not support 'Raw Send'
", 683));
    goto retry;
}
if (dvrcfg->DMediaId != 2)
{
    NMSAlert(12, 40, NUTHandle,
        InxMSG("Interface does not support ETHERNET_II
frames", 684));
    goto retry;
}
if (!CSTrcmp(dvrcfg->DShortName, DPCName))
{
    NMSAlert(12, 40, NUTHandle,
        InxMSG("Do not use DPC as the Interface", 685))
    goto retry;
}
tmpDlCfg->net_interface = interface;
}
NMSDeestroyForm(NUTHandle);
NMSPopList(NUTHandle);
return K_SELECT;
}

LONG ModifyLogInScript(FIFD *fp, int key, int *changed, NUTInfo *handle)
{
    int i;

```

```

    DlCfg_t
    LONG save;

    key = key;
    changed = changed;
    handle = handle;

    tmpDlCfg = (DlCfg_t *)fp->customData;

    if (NMSPushList(NUTHandle) == 0)
        return K_SELECT;

    NMSInitForm(NUTHandle);

    i = 0;

    NMSAppendCommentField(i, 2, MSG("wait1: ", 581), NUTHandle);
    NMSAppendStringField(i, 9, 30, NORMAL_FIELD, tmpDlCfg->wait_for_1,
        printables, F_NO_HELP, NUTHandle);

    NMSAppendCommentField(i, 40, MSG("wait Timeout 1: ", 599), NUTHandle);
    NMSAppendIntegerField(i, 56, NORMAL_FIELD, (int *)&tmpDlCfg->wait_timeo
ut_1, 2, 60, F_NO_HELP, NUTHandle);

    i++;
    NMSAppendCommentField(i, 2, MSG("Send1: ", 518), NUTHandle);
    NMSAppendStringField(i, 9, 30, NORMAL_FIELD, tmpDlCfg->send_1,
        printables, F_NO_HELP, NUTHandle);

    i++;
    NMSAppendCommentField(i, 2, MSG("wait2: ", 520), NUTHandle);
    NMSAppendStringField(i, 9, 30, NORMAL_FIELD, tmpDlCfg->wait_for_2,
        printables, F_NO_HELP, NUTHandle);

    NMSAppendCommentField(i, 40, MSG("wait Timeout 2: ", 600), NUTHandle);
    NMSAppendIntegerField(i, 56, NORMAL_FIELD, (int *)&tmpDlCfg->wait_timeo
ut_2, 2, 60, F_NO_HELP, NUTHandle);

    i++;
    NMSAppendCommentField(i, 2, MSG("Send2: ", 522), NUTHandle);
    NMSAppendStringField(i, 9, 30, NORMAL_FIELD, tmpDlCfg->send_2,
        printables, F_NO_HELP, NUTHandle);

    i++;
    NMSAppendCommentField(i, 2, MSG("wait3: ", 524), NUTHandle);
    NMSAppendStringField(i, 9, 30, NORMAL_FIELD, tmpDlCfg->wait_for_3,
        printables, F_NO_HELP, NUTHandle);

    NMSAppendCommentField(i, 40, MSG("wait Timeout 3: ", 601), NUTHandle);
    NMSAppendIntegerField(i, 56, NORMAL_FIELD, (int *)&tmpDlCfg->wait_timeo
ut_3, 2, 60, F_NO_HELP, NUTHandle);

    i++;
    NMSAppendCommentField(i, 2, MSG("Send3: ", 526), NUTHandle);
    NMSAppendStringField(i, 9, 30, NORMAL_FIELD, tmpDlCfg->send_3,
        printables, F_NO_HELP, NUTHandle);

    i++;
    NMSAppendCommentField(i, 2, MSG("wait4: ", 528), NUTHandle);
    NMSAppendStringField(i, 9, 30, NORMAL_FIELD, tmpDlCfg->wait_for_4,
        printables, F_NO_HELP, NUTHandle);

    NMSAppendCommentField(i, 40, MSG("wait Timeout 4: ", 602), NUTHandle);
    NMSAppendIntegerField(i, 56, NORMAL_FIELD, (int *)&tmpDlCfg->wait_timeo
ut_4, 2, 60, F_NO_HELP, NUTHandle);

    i++;
    NMSAppendCommentField(i, 2, MSG("Send4: ", 530), NUTHandle);
    NMSAppendStringField(i, 9, 30, NORMAL_FIELD, tmpDlCfg->send_4,

```

```
printables, F_NO_HELP, NUTHandle);
```

```
1++;
NMSAppendCommentField(1, 2, MSG("wait5: ", 532), NUTHandle);
NMSAppendStringField(1, 9, 30, NORMAL_FIELD, tmpDlocfg->wait_for_5,
    printables, F_NO_HELP, NUTHandle);
NMSAppendCommentField(1, 40, MSG("wait timeout 5: ", 603), NUTHandle);
NMSAppendIntegerField(1, 56, NORMAL_FIELD, (int *)&tmpDlocfg->wait_timeo
ut_5, 2, 60, F_NO_HELP, NUTHandle);
```

```
1++;
NMSAppendCommentField(1, 2, MSG("Send5: ", 534), NUTHandle);
NMSAppendStringField(1, 9, 30, NORMAL_FIELD, tmpDlocfg->send_5,
    printables, F_NO_HELP, NUTHandle);
```

```
1++;
NMSAppendCommentField(1, 2, MSG("wait6: ", 536), NUTHandle);
NMSAppendStringField(1, 9, 30, NORMAL_FIELD, tmpDlocfg->wait_for_6,
    printables, F_NO_HELP, NUTHandle);
NMSAppendCommentField(1, 40, MSG("wait timeout 6: ", 604), NUTHandle);
NMSAppendIntegerField(1, 56, NORMAL_FIELD, (int *)&tmpDlocfg->wait_timeo
ut_6, 2, 60, F_NO_HELP, NUTHandle);
```

```
1++;
NMSAppendCommentField(1, 2, MSG("Send6: ", 538), NUTHandle);
NMSAppendStringField(1, 9, 30, NORMAL_FIELD, tmpDlocfg->send_6,
    printables, F_NO_HELP, NUTHandle);
```

```
1++;
NMSAppendCommentField(1, 2, MSG("wait7: ", 540), NUTHandle);
NMSAppendStringField(1, 9, 30, NORMAL_FIELD, tmpDlocfg->wait_for_7,
    printables, F_NO_HELP, NUTHandle);
NMSAppendCommentField(1, 40, MSG("wait timeout 7: ", 605), NUTHandle);
NMSAppendIntegerField(1, 56, NORMAL_FIELD, (int *)&tmpDlocfg->wait_timeo
ut_7, 2, 60, F_NO_HELP, NUTHandle);
```

```
1++;
NMSAppendCommentField(1, 2, MSG("Send7: ", 542), NUTHandle);
NMSAppendStringField(1, 9, 30, NORMAL_FIELD, tmpDlocfg->send_7,
    printables, F_NO_HELP, NUTHandle);
```

```
1++;
NMSAppendCommentField(1, 2, MSG("wait8: ", 544), NUTHandle);
NMSAppendStringField(1, 9, 30, NORMAL_FIELD, tmpDlocfg->wait_for_8,
    printables, F_NO_HELP, NUTHandle);
NMSAppendCommentField(1, 40, MSG("wait timeout 8: ", 606), NUTHandle);
NMSAppendIntegerField(1, 56, NORMAL_FIELD, (int *)&tmpDlocfg->wait_timeo
ut_8, 2, 60, F_NO_HELP, NUTHandle);
```

```
1++;
NMSAppendCommentField(1, 2, MSG("Send8: ", 546), NUTHandle);
NMSAppendStringField(1, 9, 30, NORMAL_FIELD, tmpDlocfg->send_8,
    printables, F_NO_HELP, NUTHandle);
```

```
1++;
NMSAppendCommentField(1, 2, MSG("wait9: ", 548), NUTHandle);
NMSAppendStringField(1, 9, 30, NORMAL_FIELD, tmpDlocfg->wait_for_9,
    printables, F_NO_HELP, NUTHandle);
NMSAppendCommentField(1, 40, MSG("wait timeout 9: ", 607), NUTHandle);
NMSAppendIntegerField(1, 56, NORMAL_FIELD, (int *)&tmpDlocfg->wait_timeo
ut_9, 2, 60, F_NO_HELP, NUTHandle);
```

```
1++;
NMSAppendCommentField(1, 2, MSG("Send9: ", 550), NUTHandle);
NMSAppendStringField(1, 9, 30, NORMAL_FIELD, tmpDlocfg->send_9,
    printables, F_NO_HELP, NUTHandle);
```

```
1++;
```

```
save = NMSEditPortalForm(InxMSG("Login Script Editor", 582),
```

```
12, 40,
```

```
/* center line, column */
```

```
1, 78,
```

```
/* form height, width */
```

```
F_NOVERIFY, F_NO_HELP, /* Control flags, help message */
```

```
InxMSG("Save Changes?", 583),
```

```
NUTHandle);
```

```
/* Confirm message, hand
```

```
le */
```

```
if (save)
```

```
{
```

```
CMovB(tmpDlocfg->wait_for_1, Dlocfg.wait_for_1, 30*18);
```

```
CMovB(&tmpDlocfg->wait_timeout_1, &Dlocfg.wait_timeout_1, 9 * s1
```

```
zeof(LONG));
```

```
DPUpdateConfigFile();
```

```
)
```

```
NMSDestroyForm(NUTHandle);
```

```
NMSPoplist(NUTHandle);
```

```
return K_SELECT;
```

```
int NewProtocolFlag = -1;
```

```
LONG ChangeProtocol(FIELD *fp, int key, int *changed, NUTInfo *handle)
```

```
{
```

```
    Dlocfg_t *tmpDlocfg;
```

```
    LIST *listPtr, *slipList, *ppplist, *netlist;
```

```
    LONG ccode;
```

```
    LONG rcode = K_SELECT;
```

```
    key = key;
```

```
    changed = changed;
```

```
    handle = handle;
```

```
    tmpDlocfg = (Dlocfg_t *)fp->customData;
```

```
    if (NMSPushList(NUTHandle) == 0)
```

```
        return rcode;
```

```
    NMSInitList(NUTHandle, NULL);
```

```
    slipList = NMSAppendToList(MSG("Modem - SLIP", 519), (void *)OUT_
```

```
ppplist = NMSAppendToList(MSG("Modem - PPP", 521), (void *)OUT_PPP, NUTH
```

```
andle);
```

```
netlist = NMSAppendToList(MSG("LAN/WAN", 537), (void *)OUT_NETWORK, NUTH
```

```
andle);
```

```
if (tmpDlocfg->out_protocol == OUT_SLIP)
```

```
{
```

```
    listPtr = slipList;
```

```
    else if (tmpDlocfg->out_protocol == OUT_PPP)
```

```
{
```

```
        listPtr = ppplist;
```

```
    else
```

```
{
```

```
        listPtr = netlist;
```

```
    }
```

```
    ccode = NMSList(
```



```

NMSAppendMenuItem(mfcct10, InxMSG("19200", 297), 5, NUTHandle);
NMSAppendMenuItem(mfcct10, InxMSG("38400", 300), 6, NUTHandle);
NMSAppendMenuItem(mfcct10, InxMSG("57600", 304), 7, NUTHandle);
NMSAppendMenuItem(mfcct10, InxMSG("115200", 305), 8, NUTHandle);
NMSAppendMenuItem(1, 30, NORMAL_FIELD, &aud, mfcct10, NULL, NUTHandle);

1++;
NMSAppendCommentField(1, 2, MSG("Internet MTU", 309), NUTHandle);
NMSAppendIntegerField(1, 30, NORMAL_FIELD, (int *)&tmpDlocfg.mtu, 1, 65535, F_NO_HELP, NUTHandle);

1++;
NMSAppendCommentField(1, 2, MSG("Internet Inactivity(sec)", 310), NUTHandle);
NMSAppendIntegerField(1, 30, NORMAL_FIELD, (int *)&tmpDlocfg.inet_inactivity_timer, 1, 65535, F_NO_HELP, NUTHandle);

1++;
NMSAppendCommentField(1, 2, MSG("IP to IP tunneling", 537), NUTHandle);
NMSAppendBooleanField(1, 30, NORMAL_FIELD, (BYTE *)&tmpDlocfg.tunnel, F_NO_HELP, NUTHandle);
if (tmpDlocfg.tunnel)
    tunnelStr = MSG("Enabled", 627);
else
    tunnelStr = MSG("Disabled", 628);

fp = NMSAppendHotSpotField(1, 30, NORMAL_FIELD, tunnelStr,
    fp->customData = &tmpDlocfg;
    ChangeTunnel, NUTHandle);

LONG ip_address = htonl(tmpDlocfg.ip_address);
LONG gateway_address = htonl(tmpDlocfg.gateway_address);

1++;
ip0 = (ip_address) & 0xff;
ip1 = (ip_address >> 8) & 0xff;
ip2 = (ip_address >> 16) & 0xff;
ip3 = (ip_address >> 24) & 0xff;
NMSAppendCommentField(1, 2, MSG("IP Address(ISP)", 306), NUTHandle);
NMSAppendIntegerField(1, 30, NORMAL_FIELD, &ip3, 1, 255, F_NO_HELP, NUTHandle);
NMSAppendIntegerField(1, 34, NORMAL_FIELD, &ip2, 1, 255, F_NO_HELP, NUTHandle);
NMSAppendIntegerField(1, 38, NORMAL_FIELD, &ip1, 1, 255, F_NO_HELP, NUTHandle);
NMSAppendIntegerField(1, 42, NORMAL_FIELD, &ip0, 1, 255, F_NO_HELP, NUTHandle);

1++;
gw0 = (gateway_address) & 0xff;
gw1 = (gateway_address >> 8) & 0xff;
gw2 = (gateway_address >> 16) & 0xff;
gw3 = (gateway_address >> 24) & 0xff;
NMSAppendCommentField(1, 2, MSG("Hybrid Gateway Address", 307), NUTHandle);
NMSAppendIntegerField(1, 30, NORMAL_FIELD, &gw3, 1, 255, F_NO_HELP, NUTHandle);
NMSAppendIntegerField(1, 34, NORMAL_FIELD, &gw2, 1, 255, F_NO_HELP, NUTHandle);
NMSAppendIntegerField(1, 38, NORMAL_FIELD, &gw1, 1, 255, F_NO_HELP, NUTHandle);
NMSAppendIntegerField(1, 42, NORMAL_FIELD, &gw0, 1, 255, F_NO_HELP, NUTHandle);

LP, NUTHandle);
}

1++;
NMSAppendCommentField(1, 30, MSG("Slip Configuration", 552), NUTHandle);
NMSAppendCommentField(1, 2, MSG("Auto Login Enabled", 553), NUTHandle);
NMSAppendBooleanField(1, 30, NORMAL_FIELD, (BYTE *)&tmpDlocfg.auto_login, F_NO_HELP, NUTHandle);

1++;
protocolStr = MSG("Modify Auto Login Script", 535);
fp = NMSAppendHotSpotField(1, 25, NORMAL_FIELD, protocolStr,
    fp->customDataRelease = NMSFree;
    fp->customData = &tmpDlocfg;

1++;
if (tmpDlocfg.out_protocol == OUT_PPP)
{
    pppConfigStr = MSG("Modify PPP Configuration", 608);
    fp = NMSAppendHotSpotField(1, 26, NORMAL_FIELD, pppConfigStr,
        fp->customData = &tmpDlocfg;

    }
    else if (tmpDlocfg.out_protocol == OUT_NETWORK)
    {
        fp = NMSAppendHotSpotField(1, 26, NORMAL_FIELD,
            "Modify Network Configuration",
            ModifyNetConfig, NUTHandle);
        fp->customData = &tmpDlocfg;

    }

1++;
/* save =*/ NMSeditPortaForm(InxMSG("Provider Configuration Editor", 55), 12, 40,
    1, 78,
    /* form height, width */
    /* cflags */ F_NOVERIFY, F_NO_HELP,
    /* flags, help message */
    /* InxMSG("Save Changes?", 556) */ NULL, /* Confirm message, handle */
    NUTHandle);

NMSsetSortFunction(NUTHandle, oldSortFunction);
NMSdestroyForm(NUTHandle);

1++;
if (NewProtocolFlag != -1)
{
    cflags = F_FORCE;
    goto NewProtocolLoop;
}

if (!save)
    return;

tmpDlocfg.ip_address = htonl((ip0 |
    (ip1 << 8) |
    (ip2 << 16) |

```

```

tmpDlLocfg.gateway_address = htonl(gw0) |
    (gw1 << 8) |
    (gw2 << 16) |
    (gw3 << 24));

tmpDlLocfg.cinet_baud_index = baud;

if (memcmp(&dlLocfg, &tmpDlLocfg, sizeof(DlLocfg)) == 0 ||
    NMSConfirm("Save Changes?", 612), 0, 0, TRUE, NULL,
    NUTHandle, NULL) == FALSE)
    return;

/*
 * Get newest wait_for and send strings in case ModifyLogInScript()
 * changed them.
 */
CNOVB(DlLocfg.wait_for_1, tmpDlLocfg.wait_for_1, 30 * 18);
CNOVB(&dlLocfg.wait_timeout_1, &tmpDlLocfg.wait_timeout_1, 9 * sizeof(LONG));

CNOVB(&tmpDlLocfg, &dlLocfg, sizeof(DlLocfg_t));

inetChangeProtocol();

DPCUpdateConfigFile();

void ModemConfiguration(void)
{
    int i;
    int save;
    DlLocfg_t tmpDlLocfg;
    void (*oldSortFunction)(LIST *, LIST *, NUTInfo *);
    CNOVB(&dlLocfg, &tmpDlLocfg, sizeof(DlLocfg_t));
    NMSInitForm(NUTHandle);
    NMSGetListSortFunction(NUTHandle, &oldSortFunction);
    NMSSetListSortFunction(NUTHandle, NoSortHandler);

    i = 0;
    NMSAppendCommentField(i, 2, MSG("Packet Life(sec) : ", 325), NUTHandle);
    e);
    NMSAppendIntegerField(i, 24, NORMAL_FIELD, (int *)&tmpDlLocfg.packet_life
time, 1, 65535, F_NO_HELP, NUTHandle);
    i++;
    NMSAppendCommentField(i, 2, MSG("Call Setup(sec) : ", 326), NUTHandle);
    e);
    NMSAppendIntegerField(i, 24, NORMAL_FIELD, (int *)&tmpDlLocfg.call_setup_
timeout, 1, 65535, F_NO_HELP, NUTHandle);
    i++;
    NMSAppendCommentField(i, 2, MSG("Async Buffer Size : ", 212), NUTHandle);
    e);
    NMSAppendIntegerField(i, 24, NORMAL_FIELD, (int *)&tmpDlLocfg.async_buffe
r_size, 1500, 1024*100, F_NO_HELP, NUTHandle);
    i++;
    NMSAppendCommentField(i, 2, MSG("Dial Prefix : ", 329), NUTHandle);
    e);
    NMSAppendStringField(i, 24, 10, NORMAL_FIELD, tmpDlLocfg.dialout_prefix,

    i++;
    NMSAppendCommentField(i, 2, MSG("Hangup Str : ", 330), NUTHandle);
    e);
    NMSAppendStringField(i, 24, 20, NORMAL_FIELD, tmpDlLocfg.hangup_str,
    modem_control_chars, F_NO_HELP, NUTHandle);
    i++;
    NMSAppendCommentField(i, 2, MSG("Disconnect Str : ", 331), NUTHandle);
    e);
    NMSAppendStringField(i, 24, 20, NORMAL_FIELD, tmpDlLocfg.disconnect_str,
    modem_control_chars, F_NO_HELP, NUTHandle);
    i++;
    NMSAppendCommentField(i, 2, MSG("Escape Str : ", 332), NUTHandle);
    e);
    NMSAppendStringField(i, 24, 20, NORMAL_FIELD, tmpDlLocfg.escape_str,
    modem_control_chars, F_NO_HELP, NUTHandle);
    i++;
    NMSAppendCommentField(i, 2, MSG("Connect Str : ", 333), NUTHandle);
    e);
    NMSAppendStringField(i, 24, 50, NORMAL_FIELD, tmpDlLocfg.connect_str,
    modem_control_chars, F_NO_HELP, NUTHandle);
    i++;
    NMSAppendCommentField(i, 2, MSG("Init Str : ", 334), NUTHandle);
    e);
    NMSAppendStringField(i, 24, 60, NORMAL_FIELD, tmpDlLocfg.init_str,
    modem_control_chars, F_NO_HELP, NUTHandle);
    i++;
    save = NMSGetFormForm(InxMSG("Modem Configuration Editor", 513),
    12, 40,
    i, 76,
    /* form height, width */
    F_VERIFY, F_NO_HELP, /* Control flags, help m
essage */
    InxMSG("Save Changes?", 514), /* Confirm message, hand
le */
    NUTHandle);
    NMSSetListSortFunction(NUTHandle, oldSortFunction);
    NMSDestroyForm(NUTHandle);
    if (!save)
        return;
    CNOVB(&tmpDlLocfg, &dlLocfg, sizeof(DlLocfg_t));
    i++;
    if (AIOPortHandle != -1)
    {
        AIOWriteBufferSize(AIOPortHandle, DlLocfg.async_buffer_size);
        AIOWriteBufferSize(AIOPortHandle, &AIOWriteBufferSize);
        AIOWriteBufferSize = (AIOWriteBufferSize < DLOBUFSIZE) ? AIOWrite
BufferSize : DLOBUFSIZE;
        AIOWriteBufferSize = (AIOWriteBufferSize < DLOBUFSIZE) ? AIOWrite
BufferSize : DLOBUFSIZE;
    }
    DPCUpdateConfigFile();
}

```



```

*
* Description:
*   This routine is called when the modem needs to be dailed.
*
* Input:
*   nothing
*
* Output:
*   nothing
*
* Returns:
*   nothing
*
* .....
static void WriteCommPhoneNumber(void)
{
    int baudindex, i;
    char *number;

    if (DlloConn == DLO_CONN_PACKAGE)
        baudindex = DlloCfg.pdeliv_baud_index;
    else
        baudindex = DlloCfg.tinet_baud_index;

    AIOConfigurePort(AIOPortHandle,
        AIOBaudRateDefines[baudindex],
        AIO_DATA_BITS_8, AIO_STOP_BITS_1,
        AIO_PARITY_NONE,
        AIO_SOFTWARE_FLOW_CONTROL_OFF | AIO_HARDWARE_FLOW_CONTROL_ON);

    if (DlloCfg.dialout_prefix(0))
    {
        if (DlloConn == DLO_CONN_PACKAGE)
            number = DlloCfg.pdeliv_phone_num;
        else
            number = DlloCfg.tinet_phone_num;
        for (i = 0; number[i]; i++)
        {
            if (number[i] < 'A' || number[i] > 'Z')
                break;
            if (number[i] > 'Z' && number[i] < 'a')
                break;
        }
        if (i)
            SendAIData(number, i); /* Send the alpha string */
    }
    SendAIData(DlloCfg.dialout_prefix, CStrlen(DlloCfg.dialout_prefix));
    SendAIData(&number[i], CStrlen(&number[i]));
}
else
{
    if (DlloConn == DLO_CONN_PACKAGE)
        SendAIData(DlloCfg.pdeliv_phone_num, CStrlen(DlloCfg.pdeliv_phone_num));
    else
        SendAIData(DlloCfg.tinet_phone_num, CStrlen(DlloCfg.tinet_phone_num));
}

SendAIData(MSG("\r", 613), 1);
if (DlloState == DLOS_REDL)
}

}

UpdateModemStr(MSG("Modem Status: Redialing\n", 559));
else
    UpdateModemStr(MSG("Modem Status: Dialing\n", 560));
}

}

.....
ProcessDisconnect(void)
{
    Description:
    This routine is called when where attaching and we lose Carrier Detect.
}

Input:
    nothing
}

Output:
    nothing
}

Returns:
    nothing
}

.....
static void ProcessDisconnect(void)
{
    int count;

    if (DlloConn == DLO_CONN_PACKAGE)
    {
        UpdateModemStr(MSG("Modem Status: Disconnected from Package Dell\n", 237));
        count = DlloPxmItCount;
    }
    else
    {
        UpdateModemStr(MSG("Modem Status: Disconnected from Internet\n", 472));
        count = DlloXmItCount;
    }
    if (count)
    {
        WriteCommPhoneNumber();
        DlloStartTimer(DlloCfg.call_setup_timeout * 19);
        DlloState = DLOS_DIAL;
        if (DlloConn == DLO_CONN_INET)
            InetStateChange(DLOS_DIAL);
        else
        {
            DlloStartTimer(1 * 19);
            DlloState = DLOS_DISC_4;
            if (DlloConn == DLO_CONN_INET)
                InetStateChange(DLOS_DISC_4);
        }
    }
}

.....
DlloEndConn(void)
{
    Description:
}

```

This routine is terminate a modem connection.

Input: nothing
Output: nothing
Returns: nothing

void DloEndConn(void)

if (AIOPortHandle < 0)
return;

if (DloConn == DLO_CONN_PACKAGE)
DloPxmIttCount = 0;
else
DloIxmIttCount = 0;

switch(DloState)

case DLOS_INIT:
case DLOS_REDL:
case DLOS_DIAL:
case DLOS_CONN:

AIOFlushBuffers(AIOPortHandle, (AIO_FLUSH_WRITE_BUFFER |

AIO_FLUSH_READ_BUFFER));
DloState = DLOS_CONN;
StateMachine(DLOE_TIMEOUT);
break;

case DLOS_IDLE:
case DLOS_DISC_1:
case DLOS_DISC_2:
case DLOS_DISC_3:
case DLOS_DISC_4:
StateMachine(DLOE_DISCONN);
break;

_0003(void) In IDLE state, got SND event

Description:
The state machine is in the IDLE state.
We've received a SND request.

Input: nothing

Output: nothing

Returns: nothing

int DloConnected(void)

LONG extStatus = 0, chgdExtStatus;

if (AIOPortHandle < 0)
return(FALSE);

AIOGetExternalStatus(AIOPortHandle, &extStatus, &chgdExtStatus);
if (extStatus & AIO_EXTSTA_DCD)
return(TRUE);

return(FALSE);

static void _0003(void)

LONG extStatus, chgdExtStatus;

InitializeAIO();
if (AIOPortHandle < 0)

UpdateModemStr(MSG("Modem Status: ERROR - Unable to Initialize AIO", 238));
return;

if (AIOGetExternalStatus(AIOPortHandle, &extStatus, &chgdExtStatus)
|| !extStatus & AIO_EXTSTA_DCD)

AIOFlushBuffers(AIOPortHandle,
(AIO_FLUSH_WRITE_BUFFER | AIO_FLUSH_READ_BUFFER)

);

SendAIOData(DloCfg.Init_str, CString(DloCfg.Init_str));
SendAIOData(MSG("r", 614), 1);
DloStartTimer(5 * 19);

DloState = DLOS_INIT;
if (DloConn == DLO_CONN_INET)
InetStateChange(DLOS_INIT);
UpdateModemStr(MSG("Modem Status: Initializing Modem", 561));

else

if (DloConn == DLO_CONN_PACKAGE)
DloStartTimer(DloInactivityTimer);

else

DloStartTimer(DloInactivityTimer);
DloStartTimer(30 * 19);
DloStopPacketLifterTimer();

WriteCommXmitBuffer();
DloState = DLOS_CONN;
if (DloConn == DLO_CONN_INET)
InetStateChange(DLOS_CONN);

if (DloConn == DLO_CONN_PACKAGE)
UpdateModemStr(MSG("Modem Status: Connected to Package D", 562));
else
UpdateModemStr(MSG("Modem Status: Connected to Internet", 473));

if (ConnectBaudStr[0])

if (ConnectBaudStr[0])
BYTE connectStr[80];

WMprintf(connectStr, MSG("Modem Status: Connected to Internet at%.24s\n", 563), ConnectBaudStr);


```

writeCommXmitBuffer();
if (DIOConn == DLO_CONN_PACKAGE)
{
    DIOStartTimer(DIOInactivityTimer);
    UpdateModemStr(MSG("Modem Status: Connected to Package Delivery
\n", 244));
}
else
{
    DIOStartTimer(DIOInactivityTimer);
    UpdateModemStr(MSG("Modem Status: Connected to Internet
\n", 474));
    if (ConnectBaudStr[0])
    {
        BYTE connectStr[80];
        sprintf(connectStr, MSG("Modem Status: Connected to Internet
\n", 492));
        UpdateModemStr(connectStr);
    }
    else
    {
        UpdateModemStr(MSG("Modem Status: Connected to Internet
\n", 492));
    }
}
DIOStopPacketLifeTimer();
DIOState = DLOS_CONN;
if (DIOConn == DLO_CONN_INET)
    InetStateChange(DLOS_CONN);
}

.....

-0202(void)          In DIAL state, got disconnected

Description:
    The state machine is in the DIAL state.
    We got disconnected from remote side.

Input:
    nothing

Output:
    nothing

Returns:
    nothing

...../

static void _0202(void)
{
    DIOEndConn();
}

...../

-0205(void)          In DIAL state, got BUSY response from mo
dem

Description:
    The state machine is in the DIAL state.
    We've received a BUSY response from sending ATDT sequence.

```

```

Input: nothing
Output: nothing
Returns: nothing

...../

static void _0205(void)
{
    UpdateModemStr(MSG("Modem Status: Busy
\n", 245));
}

...../

from modem _0206(void)

    In DIAL state, got NODIALTONE response f

    Description:
        The state machine is in the DIAL state.
        We've received an NODIALTONE response from sending ATDT sequence

    Input: nothing
    Output: nothing
    Returns: nothing

    ...../

static void _0206(void)
{
    UpdateModemStr(MSG("Modem Status: No Dialtone
\n", 246));
    DIOStartTimer(5 * 19);
}

...../

dem _0207(void)

    In DIAL state, got RING response from mo

    Description:
        The state machine is in the DIAL state.
        We've received a RING response from sending modem ATDT sequence.

    Input: nothing
    Output: nothing
    Returns: nothing

    ...../

```

```
(
static void _0207(void)
{
    UpdateModemStr(MSG("Modem Status: Ringing!!!
\n", 247));
    /*****
    * _0208(void)          In DIAL state, got NO ANSWER response fr
    *
    * Description:
    *   The state machine is in the DIAL state.
    *   We've received a NO ANSWER response from sending ATDT sequence.
    *
    * Input:
    *   nothing
    *
    * Output:
    *   nothing
    *
    * Returns:
    *   nothing
    *****/
    static void _0208(void)
    {
        UpdateModemStr(MSG("Modem Status: No ANSWER
\n", 248));
        /*****
        * _0300(void)          In REDIAL state, got TIMEOUT
        *
        * Description:
        *   The state machine is in the REDIAL state.
        *   We timed out.
        *
        * Input:
        *   nothing
        *
        * Output:
        *   nothing
        *
        * Returns:
        *   nothing
        *****/
        static void _0300(void)
        {
            UpdateModemStr(MSG("Modem Status: Timeout - Reinitializing the modem
\n", 249));
            AIOFlushBuffers(AIOportHandle, AIO_FLUSH_WRITE_BUFFER);
            AIOFlushBuffers(AIOportHandle, AIO_FLUSH_READ_BUFFER);
            SendAIOData(DIOCfg_init_str, CStrlen(DIOCfg_init_str));
            SendAIOData(MSG("\r", 616), 1);
            DIOstartTimer(5 * 19);
            DIOstate = DLOS_INIT;
            if (DIOConn == DLO_CONN_INET)
                InetStateChange(DIOS_INIT);
        }
    }
}

/*****
* _0302(void)          In REDIAL state, got Disconnected
*
* Description:
*   The state machine is in the REDIAL state.
*   We got disconnected by the remote site.
*
* Input:
*   nothing
*
* Output:
*   nothing
*
* Returns:
*   nothing
*****/
static void _0302(void)
{
    DIOEndConn();
    /*****
    * _0104(void)          In REDIAL state, got OK response from mo
    * dem
    *
    * Description:
    *   The state machine is in the REDIAL state.
    *   We've received an OK response from sending modem init sequence.
    *
    * Input:
    *   nothing
    *
    * Output:
    *   nothing
    *
    * Returns:
    *   nothing
    *****/
    static void _0304(void)
    {
        WriteCommPhoneNumber();
        DIOstartTimer(DIOCfg_call_setup_timeout * 19);
        DIOstate = DLOS_DIAL;
        if (DIOConn == DLO_CONN_INET)
            InetStateChange(DIOS_DIAL);
    }
}

/*****
* _0400(void)          In CONNECTED state, timed out
*
* Description:
*   The state machine is in the CONNECTED state.
*   Event: We timed out due to inactivity.
*   Action: Add a 1.5 sec pre-escape delay. DLOS_DISC_1 state.
*
* Input:
*   nothing
*
* Output:
*****/
```



```

* nothing
*
* Returns:
*   nothing
*
*...../
static void _0400(void)
{
    DIOStartTimer( (1 * 19) + 9);
    DIOState = DLOS_DISC_1;
    if (DIONextConn == DIOConn)
        DIONextConn = DLO_CONN_IDLE;
    if (DIOConn == DLO_CONN_INET) {
        UpdateModemStr(MSG("Modem Status: Disconnecting from Internet
\n", 566));
        InetStateChange(DLOS_DISC_1);
    }
    else if (DIOConn == DLO_CONN_PACKAGE)
        UpdateModemStr(MSG("Modem Status: Disconnecting from Package Delivery
\n", 565));
    else
        UpdateModemStr("Modem Status: Disconnecting
\n");
}

/.....
static void _0402(void)
{
    Description:
        State: The state machine is in the CONNECT state.
        Event: We were disconnected by the remote site.
        Action: If still have data to send:
            Send connect string(ATDT
1800...), DLOS_DIAL state.
            else
                Start 1 second timer, DL
OS_DISC_4 state.
        Input:
            nothing
        Output:
            nothing
        Returns:
            nothing
        ...../
static void _0402(void)
{
    ProcessDisConnect();
}

/.....
static void _0403(void)
{
    Description:
        State: The state machine is in the CONNECTED state.
        Event: We've received a request to send data to the remote site
        Action: Extend the inactivity timer.
}

* Input:
*   nothing
*
* Output:
*   nothing
*
* Returns:
*   nothing
*
*...../
static void _0403(void)
{
    if (DIOConn == DLO_CONN_PACKAGE)
        DIOStartTimer(DIOInactivityTimer);
    else
        DIOStartTimer(DIOInactivityTimer);
}

/.....
static void _0500(void)
{
    Description:
        State: Disconnect 1 state
        Two ways to get in:
            1)
                - Inactivity timer kicke
                - Set 1.5 pre-escape tim
            2)
                - Inactivity timer kicked in
                - Set 1.5 pre-escape timer(DLO
S_DISC_1)
                - Sent escape string w/2
                - Sent hangup string w/1
                - Timed out
        Event: Intentional 1.5 or 10 second timeout.
        Action: Send escape string(+++) set 2 second timer, DLOS_DISC_2
state.
        Input:
            nothing
        Output:
            nothing
        Returns:
            nothing
        ...../
static void _0500(void)
{
    AIOFlushBuffers(AIOporthandle,
(AIO_FLUSH_WRITE_BUFFER | AIO_FLUSH_READ_BUFFER));
    if (DebugFlag)
        fputs(DIOCfg.escape_str, stdout);
    SendAIOData(DIOCfg.escape_str, CStrLen(DIOCfg.escape_str));
    DIOStartTimer(2 * 19);
}
```

```

    DloState = DLOS_DISC_2;
    if (DloConn == DLO_CONN_INET)
        InetStateChange(DLOS_DISC_2);
}

/*****
 * _0502(void)
 * Description:
 * State: Disconnect 1 state.
 *
 * Two ways to get in:
 * 1)
 * - Inactivity timer kicked
 * - Set 1.5 pre-escape timer
 * 2)
 * - Inactivity timer kicked in
 * - Set 1.5 pre-escape timer(DLO
 *   S_DISC_1)
 * - Sent escape string w/2
 * - Sent hangup string w/1
 * - Timed out
 * Event: We were disconnected by the remote site.
 * Action: If still have data to send:
 * 1800...), DLOS_DIAL state.
 *   else
 *   Start 1 second timer, DL
 * OS_DISC_4 state.
 * Input:
 *   nothing
 * Output:
 *   nothing
 * Returns:
 *   nothing
 *****/
static void _0502(void)
{
    ProcessDisconnect();
}

/*****
 * _0600(void)
 * Description:
 * State: Disconnect 2 state:
 *   - Inactivity timer kicked in
 *   - Set 1.5 pre-escape timer(DLO
 *   S_DISC_1)
 *   - Sent escape string w/2
 * second timer(DLOS_DISC_2)
 * Event: We timed out, modem didn't respond.
 * Action: Send Hangup string with 10 second timer, DLOS_DISC_3 sta
 * te.
 *****/
static void _0600(void)
{
    SendAIOData(DloCfg.hangup_str, CString(DloCfg.hangup_str));
    delay(2000);
    DloStartTimer(10 * 19);
    DloState = DLOS_DISC_3;
    if (DloConn == DLO_CONN_INET)
        InetStateChange(DLOS_DISC_3);
}

/*****
 * _0602(void)
 * Description:
 * State: Disconnect 2 state:
 *   - Inactivity timer kicked in
 *   - Set 1.5 pre-escape timer(DLO
 *   S_DISC_1)
 *   - Sent escape string w/2
 * second timer(DLOS_DISC_2)
 * Event: We got disconnected by the remote site while waiting aft
 * er
 * Action: If still have data to send:
 *   sending ***.
 *   Send connect string(ATDT
 * 1800...), DLOS_DIAL state.
 *   else
 *   Start 1 second timer, DL
 * OS_DISC_4 state.
 * Input:
 *   nothing
 * Output:
 *   nothing
 * Returns:
 *   nothing
 *****/
static void _0602(void)
{
    ProcessDisconnect();
}

/*****
 * _0604(void)
 * Description:
 * State: Disconnect 2 state:
 *   In Disconnect 2 state, got OK response f
 * rom modem
 *****/
static void _0604(void)
{
    In Disconnect 2 state, got OK response f
    rom modem
    Description:
    State: Disconnect 2 state:

```


Start 1 second timer, DL

```

OS_DISC_4 state.
    Input:  nothing
    Output: nothing
    Returns: nothing
    .....
```

```

static void _0704(void)
{
    ProcessDisconnect();
    .....
```

```

    _0800(void)          In Disconnect 4 state, got timeout
    Description:
    State:  Disconnect 4 state:
        - Inactivity timer kicked in
        - Set 1.5 pre-escape timer(DLO
S_DISC_1)
        second timer(DLOS_DISC_2)
        0 second timer(DLOS_DISC_3)
        d, set 1 second timer(DLOS_DISC_4)
        Event:  We've intentionally timed out.
        Action:  Hangup is complete and there is nothing more to send.
    Input:  nothing
    Output: nothing
    Returns: nothing
    .....
```

```

static void _0800(void)
{
    if (DLOnextConn == DLO_CONN_IDLE)
    {
        DLOConn = DLO_CONN_IDLE;
        UpdateModemStr(MSG("Modem Status: IDLE
\n", 567));
        DLOstate = DLOS_IDLE;
        InetStateChange(DLOS_IDLE);
    }
    else
    {
        DLOConn = DLOnextConn;
        DLOnextConn = DLO_CONN_IDLE;
        if (DLOConn == DLO_CONN_INET)
        {
            DLOInactivityTimer = DLOCfg.tinet_inactivity_timer * 19
    .....
```

```

        )
        )
        static _statefn statetable[DLOENUM][DLOSNUM] =
        {
            // (0) (1) (2) (3) (4) (5) (6) (7) (8)
            // IDLE INIT DIAL REDL CONN DIS1 DIS2 DIS3 DIS4 <-states
            //-----
            { 0, _0100, _0200, _0300, _0400, _0500, _0600, _0700, _0800 }, // (0) TMO <
            { 0, 0, _0201, 0, 0, 0, 0, 0, 0 }, // (1) CON
            { 0, 0, _0202, _0302, _0402, _0502, _0602, _0702, 0 }, // (2) DIS
            { _0003, 0, 0, 0, _0403, 0, 0, 0, 0 }, // (3) SND
            { 0, _0104, 0, _0304, 0, 0, _0604, _0704, 0 }, // (4) RSP
            { 0, 0, _0205, 0, 0, 0, 0, 0, 0 }, // (5) ST
            { 0, 0, 0, _0206, 0, 0, 0, 0, 0 }, // (6) T
            { 0, 0, 0, _0207, 0, 0, 0, 0, 0 }, // (7) HANG
            { 0, 0, 0, _0208, 0, 0, 0, 0, 0 }, // (8) NAN
        };

    /** The State Machine Driver ****
    static void StateMachine (int event)
    {
        #if TRACE_STATE
        char traceStr(20);
        NMSprintf(traceStr, MSG("%d%d", 610), DLOstate, event);
        fputs(traceStr, stdout);
        #endif
        if (statetable[event][DLOstate])
        {
            (*statetable[event][DLOstate]) ();
        }
    }
    .....
```

```

    DLOsend(BYTE *buffer,
            int size,
            int timeout)

    Description:
        This routine is called to send data out the modem. If we
        are connected send it. Otherwise store the message and return. The
        state machine will send the data after it has connected.
    Input:
        buffer          - data to send
        size            - size of data
        timeout         - inactivity timeout
    Output:
        nothing
    Returns:
        nothing
    .....
```

```

    int DLOsend (LPSTR buffer, int size, int timeout)
    {
        int i;
        LONG *maxBufferSize, *xmitCount;
    .....
```



```

)
else
{
    xmitBuffer = DioXmitBuffer;
    xmitCount = &DioXmitCount;
}

UpdateModemLights(1, 0, 1);

if (*xmitCount)
{
    if (DebugFlag)
    {
        printf(MSG("xmit %d bytes:\n", 486), *xmitCount);
        HexasciiDump(xmitBuffer,
            (DioConn == DIO_CONN_INET && *xmitCount > 3
            ? 32 : *xmitCount);
        )
        SendAIOData(xmitBuffer, *xmitCount);
        *xmitCount = 0;
    }
}

/*****
 *
 * DioStartPacketLifeTimer(void)
 *
 * Description:
 *   This routine starts the Packet Life timer.
 *   This timer is set to the configured packet timer(120 seconds by
 *   default) each time data is added via DioSend.
 *   DioMain will periodically decrement the timer and clear the
 *   data buffer when it reaches zero.
 *
 * Input:
 *   nothing
 *
 * Output:
 *   nothing
 *
 * Returns:
 *   nothing
 */
static void DioStartPacketLifeTimer( void )
{
    DioPacketLifeTimer = (DioCfg.Packet_Lifetime * 19 > MIN_TIMER_VALUE ?
        DioCfg.Packet_Lifetime * 19 : MIN_TIMER_VALUE);
}

/*****
 *
 * DioStopPacketLifeTimer(void)
 *
 * Description:
 *   The routine is called to stop the packet life timer.
 *
 * Input:
 *   nothing
 *
 * Output:
 *   nothing
 *
 * Returns:
 *   nothing
 */
)
}

/*****
 *
 * DioStartTimer(LONG ticks)
 *
 * Description:
 *   This routine is called to set the value of the state machine tim
 *   er.
 *   DioMain periodically decrements this timer and set the TIMEOUT
 *   event when it hits zero.
 *
 * Input:
 *   ticks
 *
 * Output:
 *   nothing
 *
 * Returns:
 *   nothing
 */
static void DioStartTimer( LONG ticks )
{
    DioTimer = (ticks > MIN_TIMER_VALUE ? ticks : MIN_TIMER_VALUE);
    #if TRACE_STATE
    {
        char traceStr[20];
        NWsprintf(traceStr, MSG("%d", 611), DioTimer);
        fputs(traceStr, stdout);
    }
    #endif
}

/*****
 *
 * DioStopTimer( void )
 *
 * Input:
 *   nothing
 *
 * Output:
 *   nothing
 *
 * Returns:
 *   nothing
 */
}

/*****
 *
 * SendAIOData(BYTE *data,
 *             int length)
 *
 * Description:
 *   The routine sends the data pointed to by data to the modem.
 *
 * Input:
 *   data
 *   length
 *
 * Output:
 *   nothing
 *
 * Returns:
 *   nothing
 */
}

```

```
void SendAIOData(BYTE *data, LONG length)
```

```
    LONG count;
    int bufferspace;
    BYTE *ptr;
    WORD state;
    LONG byteswritten;
```

```
    ptr = data;
    while (ptr < (data + length))
```

```
    {
        AIOWriteStatus(AIOPortHandle, &count, &state);
        bufferspace = AIOWriteBufferSize - count;
```

```
        if (length < bufferspace)
            bufferspace = length;
```

```
        if (bufferspace > 0)
```

```
        {
            AIOWriteData(AIOPortHandle, ptr, bufferspace, &byteswrit
```

```
ten);
```

```
        ptr += bufferspace;
```

```
        else
            ThreadSwitchWithDelay();
    }
```

```
.....
```

```
    AIOPortInfo(int portChoice,
                int *hardwareType,
                int *boardNumber,
                int *portNumber)
```

```
    Description:
```

```
    The routine returns the AIO information of the port passed in
    portChoice.
```

```
    Input:
```

```
    portChoice
```

```
    - port to return data ab
```

```
    out
```

```
    hardwareType
```

```
    - where to return hardware type
```

```
    boardNumber
```

```
    - where to return board
```

```
    number
```

```
    portNumber
```

```
    - where to return port n
```

```
    umber
```

```
    Output:
```

```
    hardwareType, boardNumber and portNumber filled in if successful
```

```
    Returns:
```

```
    0 if successful
```

```
.....
```

```
int AIOPortInfo(int portChoice,
```

```
                int *hardwareType,
```

```
                int *boardNumber,
```

```
                int *portNumber)
```

```
    {
        int ccode;
        AIOPortInfo portInfo;
        AIOPORTSEARCH portSearch;
```

```
char portOwner[130];
```

```
portInfo.returnLength = sizeof (AIOPORTINFO);
```

```
ccode = AIOGetFirstPortInfo(-1, -1, -1, &portSearch, &portInfo,
                             NULL, NULL, portOwner);
```

```
if (ccode)
    return (-1);
```

```
while (!ccode)
```

```
{
    if (portInfo.portNumber == portChoice)
```

```
{
    if (portInfo.availability == AIO_AVAILABLE_FOR_ACQUIRE)
```

```
        *hardwareType = portInfo.hardwareType;
        *boardNumber = portInfo.boardNumber;
        *portNumber = portInfo.portNumber;
        return 0;
```

```
    }
    else
```

```
    {
        return (-2);
    }
```

```
    ccode = AIOGetNextPortInfo(&portSearch, &portInfo, NULL, NULL,
                                portOwner);
    return -1;
```

```
}
```

```
.....
```

```
InitializeAIO(void)
```

```
Description:
```

```
Initialize AIO. If it didn't initialize, AIOPortHandle will
still be negative.
```

```
Input:
```

```
nothing
```

```
Output:
```

```
nothing
```

```
Returns:
```

```
nothing
```

```
.....
```

```
void InitializeAIO(void)
```

```
{
```

```
    int ccode;
```

```
    int hardware = AIO_HARDWARE_TYPE_WILDCARD;
```

```
    int port = AIO_PORT_NUMBER_WILDCARD;
```

```
    int board;
```

```
    AIODRIVERLIST dvr;
```

```
    dvr.returnLength = sizeof(AIODRIVERLIST);
```

```
    if (AIOPortHandle >= 0)
```

```
        return;
```

```
    while (AIOGetDriverList(hardware, &dvr) == AIO_SUCCESS)
        AIOBOARDLIST brd;
```



```

DlOCallBackIndex = 0;
DlOCallBackEscape = 0;
DlOCallBackStarted = 0;
return(0);
}

/*.....*/

ValidPacket(BYTE *buf_to_rx,
            int *len_to_rx)
{
    Description:
        This routine validates a response from the modem. It checks the
        length, checksum, opcode and status.

    Input:
        buf_to_rx          - pointer to the response message
        len_to_rx          - pointer to the total length of the message

    Output:
        len_to_rx          - changed to the real size of the message
        *the header        - withou

    Returns:
        TRUE if packet is valid
        FALSE if packet is invalid
}

/*.....*/

int ValidExplicitPacket(BYTE *buf_to_rx, int *len_to_rx)
{
    LROESpKt_t *msg;
    WORD frameCrc = 0;
    WORD slpCrc = 0;
    WORD frameLen = 0;
    WORD crcVal = 0xffff;

    msg = (LROESpKt_t *)buf_to_rx;
    frameLen = (msg->length) & 0x7fff;

    if ((*len_to_rx - sizeof(WORD)) == frameLen)
    {
        MOV(Buf_to_rx[frameLen], (BYTE *)&frameCrc, sizeof(frameCrc));
        slpCrc = calcCrc(crcVal, buf_to_rx, frameLen);
        if (slpCrc == frameCrc)
        {
            return(1);
        }
    }
    return(0);
}

int ValidPacket(BYTE *buf_to_rx, int *len_to_rx)
{
    unsigned short frameCrc=0; // CRC value contained in the frame
    unsigned short frameLen=0; // Length value contained in the frame
    unsigned short slpCrc=0; // CRC returned from calculation
    LROASyncMsg_t *msg;

    DcauResponse_t *d_msg;
    int status = FALSE;

    msg = (LROASyncMsg_t *)buf_to_rx;
    // extract the frame length contained in the first 2 bytes of the frame
    frameLen = msg->header.length;
    frameLen &= 0x7fff;

    if ((*len_to_rx - sizeof(unsigned short)) == frameLen)
    {
        // since slpCrc includes CRC
        // extract the crc contained in the last 2 bytes of the frame
        frameCrc = msg->data[frameLen - LRO_ASYNC_HDR_SIZE];
        frameCrc += msg->data[frameLen - LRO_ASYNC_HDR_SIZE + 1] * 256;

        slpCrc = calcCrc (INITCRC, (unsigned char *) buf_to_rx, frameLen);
        if (slpCrc == frameCrc)
        {
            d_msg = (DcauResponse_t *) (msg->data);
            if (d_msg->opcode == 1 & d_msg->status == 0)
            {
                *len_to_rx = frameLen - RETURN_POINT;
                status = TRUE;
            }
        }
    }
    return status;
}

/*.....*/

DlOCallBackRead(void)
{
    Description:
        This routine reads as many bytes as it can from the modem
        on behalf of the process that scheduled a receive thru
        DlOScheduReReceive(). It's called by the main DLO thread
        as long as a call back is scheduled(DlOCallBack != 0) and
        the modem has sent the previous request. All Slp specific
        characters are stripped and all Slp escape characters are
        converted back to ASCII. If the end of the message is hit,
        call the processes event routine with the message.

    Input:
        nothing

    Output:
        nothing

    Returns:
        nothing
}

/*.....*/

static void DlOCallBackRead()
{
    BYTE value;

    for(;;)
    {
        if (DlOReceiv(evalue, 1) == 0)
            break;
        if (DebugFlag)
            putchar(value);
    }
}

```



```

void DIOmain(void *parm)
{
    LONG curticks, delta;
    LONG count, bytesRead;
    WORD state;
    BYTE value;
    int event, eventLen;
    char *pPtrAtBegin, *pPtrAtEnd;
    LONG extStatus;

    parm = parm;

    DIOlastKnownTickCount = GetCurrentTime();

    while(!ExitingFlag)
    {
        delay(1000 / 6);

        count = 0;
        bytesRead = 0;
        if (AIOGetPortStatus(AIOPortHandle,
            &count, /* write count */
            0, /* write status */
            &bytesRead, /* read count */
            0, /* read status */
            &extStatus,
            0) == 0)
        {
            extStatus &= AIO_EXTSTA_DCD;
            if (extStatus != DIOlastDCD)
            {
                if (!extStatus)
                {
                    StateMachine(DIOE_DISCONN);
                    DIOlastDCD = extStatus;
                }
            }
            UpdateModemLights(count, bytesRead, DIOlastDCD);
            if (DIOstate == DIOE_IDLE)
            {
                if (DIOEPMitCount)
                {
                    DIOConn = DIO_CONN_PACKAGE;
                    StateMachine(DIOE_SEND);
                }
                else if (DIOIMitCount)
                {
                    DIOConn = DIO_CONN_INET;
                    StateMachine(DIOE_SEND);
                }
            }
            curticks = GetCurrentTime();
            if (curticks < DIOlastKnownTickCount)
            {
                delta = curticks + (0xffffffff - DIOlastKnownTickCount);
            }
            else
            {
                delta = curticks - DIOlastKnownTickCount;
            }
            DIOlastKnownTickCount = curticks;
        }
    }
}

if (DIOStats && curticks > DPCNextRegistrationCheck)
{
    char buf[16];
    LONG hw;
    double key;
    CustomVars* customPtr = (CustomVars*)(ADIOStats->CustomVaria
        LONG rxFreq = customPtr->CustomVariable[1] / 10;

    DPCSetMaxConnections((LONG*)&key);
    if (strcmp(DIOCfg_base_license, "Helius, Inc.", 8) == 0)
        DPCNextRegistrationCheck = (LONG)(-1);
    goto skipRegistrationCheck;
}

/* get hardware serial number */
*buf = 0;
DIOGetSN(buf);
hw = strtoul(buf, 0, 10);
if (hw == 0) {
    ConsolePrintf("\r\nDPCAgent: could not obtain hardware
        serial number of DPC card\n");
    goto disabledDPCAgent;
}

/* compute the registration key */
if (DebugFlag == 0x98) {
    printf("\rRegCheck: %e\n", key);
    HexasciiDump((void*)&key, sizeof(key));
}
key *= hw + DPC_IP_Address;
if (DebugFlag == 0x98) {
    printf("\rRegCheck: %e %d %08x\n",
        key, hw, DPC_IP_Address);
    HexasciiDump((void*)&key, sizeof(key));
}
if (key == *(double*)&DIOCfg_key)
    DPCNextRegistrationCheck = curticks + 131072; /*
else
{
    ConsolePrintf("\r\nDPCAgent: detected a bad regi
        stration key\n");
    disabledDPCAgent;
    DPCMaxConnections = 3;
    RingTheBell();
    if (DPCNextRegistrationCheck) {
        DIOCfg_gateway_address = 0;
        StateMachine(DIOE_TIMEOUT);
        DPCNextRegistrationCheck = curticks + 2185; /*
    }
    else
        DPCNextRegistrationCheck = curticks + 131072;
}
/* 120 minutes */
if ((DPCNextRegistrationCheck & 0xffffffff) == 0xffffffff
    0)
{
    skipRegistrationCheck;
    if (AIOPortHandle >= 0)

```

```

    for (;;)
    {
        if (AIOReadStatus(AIOPortHandle, &count, &state)
            break;
        AIOReadData(AIOPortHandle, &value, 1, &bytesRead);
        if (DebugFlag && DIOState != DIOS_CONN)
            putchar(value);
        if (DIORecvCount < DIOBUFSIZE)
        {
            DIORecvBuffer[DIORecvIndex] = value;
            DIORecvIndex++;
            if (DIORecvIndex >= DIOBUFSIZE)
                DIORecvBuffer[DIORecvIndex] = 0;
            DIORecvCount++;
        }
        if (value != '\n' && value != '\r')
        {
            memcpy(DIOCommandBuffer, DIOCommandBuffer
                + 1, DIOCMDBUFSIZE - 1);
            DIOCommandBuffer[DIOCMDBUFSIZE - 1] = va
                lue;
            if (DIOCommandIndex < DIOCMDBUFSIZE)
                DIOCommandIndex++;
        }
        if (value != '\r')
            continue;
        pPtrAtBegin = DIOCommandBuffer + DIOCMDBUFSIZE -
            eventLen;
        if (strlen(DIOCompareString(DIOE_CONNECT))
            if (strcmp(pPtrAtBegin, DIOCompareString(DIOE_C
                ONNECT), eventLen) == 0)
        {
            memcpy(ConnectBaudStr, pPtrAtBegin + eve
                ntLen, DIOCommandIndex - eventLen);
            ConnectBaudStr[DIOCommandIndex - eventLe
                n + 1] = 0;
            DIOFlushReceive();
            StateMachine(DIOE_CONNECT);
        }
        else
        {
            for (event = 0; event < DIOENUM; event++)
            {
                if (event == DIOE_CONNECT)
                    continue;
                eventLen = CString(DIOCompareStr
                    ing(event));
                if (eventLen == 0)
                    continue;
                pPtrAtEnd = DIOCommandBuffer + D
                    IOCMDBUFSIZE - eventLen;
                if (strcmp(pPtrAtEnd, DIOCompar
                    eString(event), eventLen) != 0)
                    continue;
                DIOFlushReceive();
                StateMachine(event);
            }
        }
    }
}

/* Check Packet lifetime timer.
 * This timer is initialized when data is added to the
 * data buffer via DIOSend.
 * If the timer expires, clear the data buffer.
 */
if (DIOPacketLifetimeTimer)
{
    if (delta >= DIOPacketLifetimeTimer)
    {
        DIOPacketLifetimeTimer = 0;
        if (DIOConn == DIO_CONN_PACKAGE)
            DIOPacketCount = 0;
        else
            DIOXmtCount = 0;
    }
    else
        DIOPacketLifetimeTimer -= delta;
}

/* Check the Receive Call Back timer.
 * If it times out, call call back routine with timeout
 */
if (DIOCallBack)
{
    if (DIOCallBackWait)
    {
        /* Wait until the send request is sent b
        if (DIOEmpty())
        {
            DIOCallBackWait = 0;
        }
        else
        {
            before starting timer */
        }
    }
}

```

Thu Jul 17 14:46:11 1997

dloc

Page 55

```

    if (delta >= DlocallBackTimeout)
    {
        /* Timeout!!! Call routine with
        DlocallBackWait = 0;
        DlocallBack(DlocallBackBuffer, D
        DlocallBack = 0;
        }
        else
            DlocallBackTimeout -= delta;
        /* Read any available modem characters f
        DlocallBackRead();
    }
}

if (AIOPortHandle >= 0)
    AIOReleasePort(AIOPortHandle);
DPCModemPID = 0;
}
```

BEGIN MANUAL_ENTRY (History, DPC/HISTORY)

DirectPC Driver Code for Netware 386.
This driver must be loaded after MSX.NLM and ETHERNET.NLM.

Written by: DFS
Date: November, 1995

History Log: 0.0 - First demo version
version 0.2 - (3-15-96) Put SignalQuality and RxFreq into stats table
version 0.3 - (4-09-96) Fixed CloseChannel so that DPCAGENT.NLM
can be properly reloaded.
version 0.4 - (4-16-96) Added code to send turbo internet packets
up to the ethernet RPacketOffset 2 bytes into
envelope in order to build 14 byte ethernet packets
version 0.5 - (4-24-96) Adjusted RPacketOffset 2 bytes into
envelope in order to build 14 byte ethernet packets
over DPC's 12 byte header of phase 2 -DEBUG on the command line
version 0.6 - (5-16-96) Completion of phase 2 -DEBUG on the command line
version 0.6 - (6-11-96) Debug screen only active if -DEBUG on the command line
version 1.00 - (6-11-96) Debbug screen only active if -DEBUG on the command line
version 1.01 - (6-17-96) Added check to end of DriverInit for locked adapter
Added Agent Register call to DriverManagement so that
we can call agent when we're being removed, allowing
agent to go into idle state until we're loaded again.

version 1.02 - (6-20-96) Added freq= to command line.
version 1.03 - (6-24-96) Fixed packet too big bug.
version 1.04 - (6-25-96) Fixed packet too big bug.
version 1.05 - (6-28-96) Really Fixed packet too big bug.
version 1.06 - (7-10-96) Enhanced dpcagent.nlm
Reduced MUIDMaximumSize so Netscape wouldn't overflow
when tunneling enabled.
version 1.07 - (7-13-96) Stopped calling SendComplete from DriverSend
Added TxFreeCount to 32
Bumped TxFreeCount to 32
version 1.09a - (8-28-96) Reduced max packet size if we couldn't hook bind event.
version 1.20 - 2-12-96 Returned call to DataLinkFrame to DPCXFrame
Changed copyright string

END MANUAL_ENTRY

name DPC
title DirectPC LAN Driver (HSM Version)
subttl -- Structures and Equate Values --
page
include driver.inc -- DirectPC Specific Equates --

Equates.

subttl -- DirectPC Specific Structures --

page
MLID_MAJOR_VERSION equ 01
MLID_MINOR_VERSION equ 20
TRUE equ 01
FALSE equ 00
TIMESTAMP equ FALSE
TIMESTAMP_BUFFER_SIZE equ (256 * 4)
INT_3 equ 3
EVENT_PROTOCOL_BIND equ 34
EVENT_PROTOCOL_UNBIND equ 33

INT_3 equ 33
EVENT_PROTOCOL_BIND equ 34
EVENT_PROTOCOL_UNBIND equ 33

STAT_BUSY equ 01
STAT_ERROR equ 02
STAT_RX_INT equ 04h
STAT_NO_RBD equ 08h
STAT_RX_RBD equ 10h
STAT_NO_INT equ 20h
STAT_SOUTPORT equ 20h

STAT_BUSY equ 01
STAT_ERROR equ 02
STAT_RX_INT equ 04h
STAT_NO_RBD equ 08h
STAT_RX_RBD equ 10h
STAT_NO_INT equ 20h
STAT_SOUTPORT equ 20h

STAT_BUSY equ 01
STAT_ERROR equ 02
STAT_RX_INT equ 04h
STAT_NO_RBD equ 08h
STAT_RX_RBD equ 10h
STAT_NO_INT equ 20h
STAT_SOUTPORT equ 20h

STAT_BUSY equ 01
STAT_ERROR equ 02
STAT_RX_INT equ 04h
STAT_NO_RBD equ 08h
STAT_RX_RBD equ 10h
STAT_NO_INT equ 20h
STAT_SOUTPORT equ 20h

STAT_BUSY equ 01
STAT_ERROR equ 02
STAT_RX_INT equ 04h
STAT_NO_RBD equ 08h
STAT_RX_RBD equ 10h
STAT_NO_INT equ 20h
STAT_SOUTPORT equ 20h

STAT_BUSY equ 01
STAT_ERROR equ 02
STAT_RX_INT equ 04h
STAT_NO_RBD equ 08h
STAT_RX_RBD equ 10h
STAT_NO_INT equ 20h
STAT_SOUTPORT equ 20h

STAT_BUSY equ 01
STAT_ERROR equ 02
STAT_RX_INT equ 04h
STAT_NO_RBD equ 08h
STAT_RX_RBD equ 10h
STAT_NO_INT equ 20h
STAT_SOUTPORT equ 20h

STAT_BUSY equ 01
STAT_ERROR equ 02
STAT_RX_INT equ 04h
STAT_NO_RBD equ 08h
STAT_RX_RBD equ 10h
STAT_NO_INT equ 20h
STAT_SOUTPORT equ 20h

STAT_BUSY equ 01
STAT_ERROR equ 02
STAT_RX_INT equ 04h
STAT_NO_RBD equ 08h
STAT_RX_RBD equ 10h
STAT_NO_INT equ 20h
STAT_SOUTPORT equ 20h

; RBD status bits.

```
FRAMING_ERR equ 0001h ; Framing error
CRC_ERR equ 0002h ; CRC error
ABORT equ 0004h ; Abort
ALIGN_ERR equ 0008h ; Alignment error
DES_ERR equ 0010h ; DES Error
SOF_BIT equ 0020h ; Start of Frame(not used)
EOF_BIT equ 0040h ; End of Frame
OVERRUN_ERR equ 0080h ; Frame Overrun bit
EMPTY equ 8000h ; if reset, buffer may be used
STATUS_ERROR equ FRAMING_ERR OR CRC_ERR OR ABORT OR ALIGN_ERR OR
DES_ERR OR OVERRUN_ERR
```

DebugMessage macro mask, message

local DebugMessageExit

```
test DebugMask, mask
je DebugMessageExit
```

```
push eax
push ecx
push edx
```

```
push offset message
push DPCScreen
call OutputToScreen
lea esp, [esp + (2 * 4)]
```

```
pop edx
pop ecx
pop eax
```

DebugMessageExit:

endm

DebugMessage1 macro mask, message, parm0

local DebugMessageExit

```
test DebugMask, mask
je DebugMessageExit
```

```
push eax
push ecx
push edx
```

```
push parm0
push offset message
push DPCScreen
call OutputToScreen
lea esp, [esp + (3 * 4)]
```

```
pop edx
pop ecx
pop eax
```

DebugMessageExit:

endm

DebugMessage2 macro mask, message, parm0, parm1

local DebugMessageExit

```
test DebugMask, mask
je DebugMessageExit
```

```
push eax
push ecx
push edx
```

```
push parm1
push parm0
push offset message
push DPCScreen
call OutputToScreen
lea esp, [esp + (4 * 4)]
```

```
pop edx
pop ecx
pop eax
```

DebugMessageExit:

endm

DebugMessage3 macro mask, message, parm0, parm1, parm2

local DebugMessageExit

```
test DebugMask, mask
je DebugMessageExit
```

```
push eax
push ecx
push edx
```

```
push parm2
push parm1
push parm0
push offset message
push DPCScreen
call OutputToScreen
lea esp, [esp + (5 * 4)]
```

```
pop edx
pop ecx
pop eax
```

DebugMessageExit:

endm

DebugMessage4 macro mask, message, parm0, parm1, parm2, parm3

local DebugMessageExit

```
test DebugMask, mask
je DebugMessageExit
```

```
push eax
push ecx
push edx
```

```
push parm3
push parm2
push parm1
push parm0
push offset message
push DPCScreen
call OutputToScreen
lea esp, [esp + (6 * 4)]
```

```
pop edx
pop ecx
pop eax
```


DebugMessageExit:

endm

DebugMessages macro mask, message, parm0, parm1, parm2, parm3, parm4

local DebugMessageExit

test DebugMask, mask
je DebugMessageExitpush eax
push ecx
push edxpush parm4
push parm3
push parm2
push parm1
push parm0
push offset message
push DPCScreen
call OutputToScreen
lea esp, [esp + (7 * 4)]pop edx
pop ecx
pop eax

DebugMessageExit:

endm

DebugMessages macro mask, message, parm0, parm1, parm2, parm3, parm4, parm5

local DebugMessageExit

test DebugMask, mask
je DebugMessageExitpush eax
push ecx
push edxxor eax, eax
mov al, parm5
push eax
mov al, parm4
push eax
mov al, parm3
push eax
mov al, parm2
push eax
mov al, parm1
push eax
mov al, parm0
push eax
push offset message
push DPCScreen
call OutputToScreen
lea esp, [esp + (8 * 4)]pop edx
pop ecx
pop eax

DebugMessageExit:

endm

SLOW

push eax
in al, 61h
in al, 61h
in al, 61h
pop eax
endm

BufferStruct struct

BufPtr
DataSize
BufferStruct ends

LAST_EOF equ 80000000h ; or with DataSize

MAX_APPL_RBD equ 350
MAX_CHAN equ 10
RBD_NOT_USED equ -1
MAX_CONF_ADDR equ 8
MAX_ADDR equ 16

RX_CNTL struct

RxChannel

RxESR

RX_CNTL ends

RX_FLAG_STATS_ONLY equ 1

ChannelConfig struct

CfgChannel dd 0 ; Channel ID

CfgESR dd 0 ; max # of buffers used

CfgNumAddresses dd 0 ; Number of filter addrs

CfgAddress db 6 dup (0) ; that is to follow

CfgGroupKey db 8 dup (0) ; list of addrs

CfgElementKey db 8 dup (0) ; Group keys

ChannelConfig ends ; Element keys

FilterStruct struct

FilterAddress db 6 dup (0) ; 48 bit address

FilterChannel db 2 dup (0) ; Align next field

FilterCmdIdxIndex dd 0 ; Channel ID

FilterTotalCount dd 0 ; RISC SW cmd blk index

FilterSeqCount dd 0 ; # of frames rxed

FilterSeqNum dd 0 ; # of out-of-seq frames

FilterStruct ends ; next seq expected

EblkStruct struct

EblkCmd dw 0 ;

EblkPortID dw 0 ;

EblkNotUsed dw 0 ;

EblkAddress db 6 dup (0) ;

EblkGroupKey db 8 dup (0) ;

EblkElemKey db 8 dup (0) ;

EblkStruct ends ;


```

; Parameters required by ParseDriverParameters.
;
;*****
DebugText      db      '-DEBUG'      ; Break into debugger keyword.
DebugTextLen   equ     '$-DebugText OR T_STRING
FreqText       db      'FREQ'       ; Set RxFreq
FreqTextLen    equ     '$-FreqText OR T_NUMBER
               dd      0             ; min
               dd      10000         ; max
               dd      0
               dd      0

DPCKeywordText dd      DebugText    ; First Keyword.
               dd      FreqText     ; Second Keyword.
DPCTextLen     dd      DebugTextLen ; First Keywords length.
               dd      FreqTextLen  ; Second Keywords length.

DPCProcessKeywordTab
               dd      DebugRoutine ; First Keyword routine.
               dd      FreqRoutine  ; Second Keyword routine.

Global RxFreq
               dd      DEFAULT_RX_FREQ

IOPort0Data    dd      12, 100h, 140h, 180h, 1c0h
               dd      200h, 240h, 280h, 2c0h
               dd      300h, 340h, 380h, 3c0h
Interrupt0Data dd      8, 3, 4, 5, 9, 10, 11, 12, 15

AdapterOptions AdapterOptionDefinitionStructure < .IOPort0Data, .Interrupt0Data>

SearchTbl
               dd      MINUS_OFFSET
               dd      ZERO_OFFSET
               dd      PLUS_OFFSET

StateTbl
               dd      offset InitState
               dd      offset SynchPrmState
               dd      offset AcqPDelayState
               dd      offset AcqPDSRate
               dd      offset EnableBTRState
               dd      offset StartSearchForFECState
               dd      offset CheckForFECState
               dd      offset SetOtherModeState
               dd      offset TrackingState
               dd      offset PointingAcquisitionState
               dd      offset PointingTrackingState
               dd      offset HaltState

MajorVer      dd      0
MinorVer      dd      0
Year          dd      0
Month         dd      0
Day           dd      0
ScreenRtag    dd      0
DPCScreen     dd      0

OurAdapterDataSpace dd      0

;*****
; MIPX code.
; Contains array MIPSCode and dword MIPSCodeSize.
;*****
include mips.dat
;*****
; Message equates.
;*****
CR            equ     13
LF            equ     10

;*****
; Run-time error message strings.
;*****
; DPC error messages.
;*****
TransmitTimeoutMessage db      66, 00, 'The cable might be disconnected on the
Board.', CR, LF, 0
DMANotCompleteError   db      200, 00, 'The board's DMA did not complete the w
rite.', CR, LF, 0
;*****
; InitNIC error message strings.
;*****
ErrorAllocatingRtagMessage db      72, 00, 'A resource tag is unavailable.'
, CR, LF, 0
NICNotInSlotMessage      db      50, 00, 'The board cannot be found.', CR, LF, 0
PortFailMessage          db      54, 00, 'The board did not respond to the initia
lization command.', CR, LF, 0
NICIn8BitSlotMessage     db      223, 00, 'The board must be placed in a 16-bit s
lot.', CR, LF, 0
NICInNE1000Message       db      224, 00, 'This board is configured as an NE1000.
', CR, LF, 0
NICBadConfiguration      db      80, 00, 'The board could not be configured.', CR
, LF, 0
BadISRMsg                db      207, 00, 'Unable to set ISR for test ISR.', CR
, LF, 0
NoInterruptMsg           db      208, 00, 'Interrupt is non functional. Try using another
interrupt.', CR, LF, 0
ErrorAllocatingMemoryMsg db      73, 00, 'Unable to allocate memory.', CR
, LF, 0
MSGIOSetFailed           db      200, 00, 'Unable to set the spare output. Choose
another base I/O port.', CR, LF, 0
MSGIOClearFailed         db      201, 00, 'Unable to clear the spare output. Choo
se another base I/O port.', CR, LF, 0
MSGBadRAM                db      51, 00, 'Board RAM failed the memory test.', CR
, LF, 0
LockedAdapterMsg         db      202, 00, 'Unable to start the adapter. Power dow
n system and reboot.', CR, LF, 0
TimerDesc               db      "DPC timer", 0
EventRtagMessage         db      "DPC Events", 0
InterruptRtagMessage     db      "DPC Test ISR", 0
LastDebugMessage         dd      ?
;*****
```


... . Destroyed
EDI

Findings:

Note: Interrupts preserved.

Remarks: This routine is called by the ethernet media module. It can be called at process or interrupt time.

See Also: [ETHERTSM\ETHERTSMaddmulticastaddress](#)
[ETHERTSM\ETHERTSMdeleteemulticastaddress](#)
[ETHERTSM\ETHERTSMupdateemulticast](#)

END_MANUAL_ENTRY

ServerManagement proc

First reset Multicast Address Registers.

```

cmp     dword ptr [esi], RProtocolID+0, 'TRND' ; ID=0 == 'DRCT'?
jne     BadParameterExit                       ; Jump if not
jne     BadParameterExit                       ; ID=4 == 'PC'?
jne     BadParameterExit                       ; Jump if not

```

```
movzx    ecx, [esi].LogicalID    ; ECX = Function
cmp      ecx, LastManagementFunction ; Supported?
ja       BadParametersExit
jmp      ManagementJumpTable[ecx * 4]
```

dParametersExit:

```
mov     eax, BadParameters
ret
```

```

            endp
            subctl -- SignText --
            page

```

BEGIN_MANUAL_ENTRY(Signtext, DPC/API/SIGNTXT)

Name: **SignText**

Description: This routine is called by DriverManagement to sign the text passed in.

On Entry:

On Entry:	
EAX	N/A
EBX	Frame Data Space
ECX	N/A
EDX	N/A
EBP	Adapter Data Space
ESI	ECB
EDI	N/A

Note: Interrupts are in any state.

On Return:

On Return:	EAX	Destroyed
	EBX	Preserved
	ECX	Destroyed
	EDX	Destroyed

EBP	Preserved
ESI	Preserved
EDI	Preserved

Flags:

Note: Interrupts preserved.

This routine is called by DriverManagement. It is called at process time.

TRY

```
public SignText  
SignText  
proc
```

```
mov     esi, [esi].PacketOffset
mov     edi, [esi+0]
or      edi, edi
je      SignTextError

cmp     dword ptr [esi+8], 0
je      SignTextError
```

```
cli
mov     edx, [ebp].IOMsgRamPer
mov     eax, 18800h / 2
out     dx, ax
mov     ecx, [esi+4]
shr     ecx, 1
mov     edx, [ebp].IOMsgRam
inc     ecx
```

SendStringLoop:

```
mov     ax, [edi]
out     dx, ax
add     edi, 2
dec     ecx
jne     SendStringLoop
sti
```

```
mov     edx, [ebp].IOMsgRamPtr
mov     eax, (18100h + (14 * size Eb1kStruct) + 4) / 2
out     dx, ax
```

```
mov     edx, [ebp].IOMsgRam
mov     eax, [esi + 4]
out     dx, ax
```

```
mov     edx, [ebp].IOMsgRamPtr
mov     eax, (18100h + (14 * size Eb1kStruct)) / 2
dx, ax
```

```
mov     edx, [ebp].IOMsgRam
mov     eax, 10h
out     dx, ax
```

```
sti
mov     ecx, 10
mov     edi, 10h
```

WaitForCommandDone:

```
push    ecx
mov     eax, (ebp).TimerTag
push    eax
push    2
call    DelayMyself
add     esp, (2 * 4)
pop     ecx

cld
mov     edx, [ebp].IOMsgRamPcr
mov     eax, (18100h + (14 * size Eb1kStruct)) / 2
out     dx, ax

mov     edx, [ebp].IOMsgRam
in      ax, dx
sti
cmp     ax, 0eeh
je      SignTextError
jne     WaitForCommandDone

dec     ecx
jne     WaitForCommandDone
```

ReadyToGetSignature:

```
mov     edi, [esi+8]
cld
```

```
mov     edx, [ebp].IOMsgRamPcr
mov     eax, 18790h / 2
out     dx, ax
```

```
mov     ecx, 4
mov     edx, [ebp].IOMsgRam
in      ax, dx
mov     [edi], ax
add     edi, 2
dec     ecx
jne     GetSignatureLoop
```

```
sti
mov     dword ptr [esi+4], 8
xor     eax, eax
ret
```

SignTextError:

```
mov     eax, -1
ret
```

SignText

```
subttl  -- GetSN --
page
```

BEGIN_MANUAL_ENTRY (GetSN, DPC/API/GETSN)

Name: GetSN

```
Description: This routine is called by DriverManagement to
return the adapters serial number.
```

On Entry:

EAX	N/A
EBX	Frame Data Space
ECX	N/A
EDX	N/A
EBP	Adapter Data Space
ESI	ECB
EDI	N/A

Note: Interrupts are in any state.

On Return:

EAX	Destroyed
EBX	Preserved
ECX	Destroyed
EDX	Destroyed
EBP	Preserved
ESI	Preserved
EDI	Preserved

Flags:

Note: Interrupts preserved.

Remarks:

This routine is called by DriverManagement.
It is called at process time.

See Also:

END_MANUAL_ENTRY

```
public GetSN
GetSN proc
```

```
mov     esi, [esi].RPacketOffset
```

```
cld
mov     edx, [ebp].IOMsgRamPcr
mov     eax, 18760h / 2
out     dx, ax
```

```
mov     ecx, 3
mov     edx, [ebp].IOMsgRam
in      ax, dx
mov     [esi], ax
add     esi, 2
dec     ecx
jne     GetSNLoop
```

```
sti
mov     [esi], cl
ret
```

GetSN

```
subttl  -- CloseChannel --
page
```

BEGIN_MANUAL_ENTRY (CloseChannel, DPC/API/CLSCHAN)

Name: CloseChannel

```
Description: This routine is called by DriverManagement to
```


close the specified channel.

On Entry: EAX N/A
EBX Frame Data Space
ECX N/A
EDX N/A
EBP Adapter Data Space
ESI ECB
EDI N/A

Note: Interrupts are in any state.

On Return: EAX Destroyed
EBX Preserved
ECX Destroyed
EDX Destroyed
EBP Preserved
ESI Preserved
EDI Preserved

Flags:

Note: Interrupts preserved.

Remarks: This routine is called by DriverManagement.
It is called at process time.

See Also:

END_MANUAL_ENTRY

public CloseChannel
CloseChannel proc

mov esi, [esi].RPacketOffset ; ESI -> config structure
mov esi, [esi] ; ESI = channel

cmp esi, MAX_CHAN
ja CloseChannelError
lea edi, [ebp].RxControl[esi*8]
cmp [edi].RxChannel, RBD_NOT_USED
je CloseChannelError

mov [edi].RxChannel, RBD_NOT_USED
mov [edi].RxBESR, 0

mov ecx, MAX_ADDR
lea edi, [ebp].Filter
CloseChannelCloseFilterLoop:
cmp [edi].FilterChannel, esi
jne CloseChannelCloseFilterNext

mov [edi].FilterChannel, RBD_NOT_USED
mov eax, [edi].FilterCmdBkIndex
mov [ebp].EdtBusyFlags[esi], 0

push ecx
mov ecx, size EdtStruct
mul ecx
mov edx, [ebp].IOMsgRamPtr
add eax, 18100h
shr eax, 1
push eax
add eax, 3

out dx, ax

mov edx, [ebp].IOMsgRam
xor eax, eax
out dx, ax
out dx, ax
out dx, ax

pop eax
pop ecx
mov edx, [ebp].IOMsgRamPtr
out dx, ax
mov edx, [ebp].IOMsgRam
mov eax, 1
out dx, ax

CloseChannelCloseFilterNext:
add edi, size FilterStruct
dec ecx
jne CloseChannelCloseFilterLoop

xor eax, eax
ret

CloseChannelError:
mov eax, -1
ret

CloseChannel endp
subttl -- AddAddress --
page

BEGIN_MANUAL_ENTRY(AddAddress, DPC/API/ADDADRS)

Name: AddAddress

Description: This routine is called by DriverManagement to
add the address passed in.

On Entry: EAX N/A
EBX Frame Data Space
ECX N/A
EDX N/A
EBP Adapter Data Space
ESI ECB
EDI N/A

Note: Interrupts are in any state.

On Return: EAX Destroyed
EBX Preserved
ECX Destroyed
EDX Destroyed
EBP Preserved
ESI Preserved
EDI Preserved

Flags:

Note: Interrupts preserved.

Remarks: This routine is called by DriverManagement.
It is called at process time.

; See Also:

; END_MANUAL_ENTRY

public AddAddress
proc

```
mov esi, [esi].RPacketOffset
mov eax, [esi].CfgChannel
cmp eax, MAX_CHAN
ja OpenChannelError

lea edi, [ebp].RxControl[edx*8]
cmp [edi].RxChannel, RBD_NOT_USED
je OpenChannelError
```

; Fall thru to AddAddr

AddAddress endp
subttl -- AddAddr --
page

; BEGIN_MANUAL_ENTRY(AddAddr, DPC/API/ADDADDR)

; Name: AddAddr

; Description: This routine is called by AddAddress and OpenChannel
; to add the address to the adapter.

; On Entry:

```
EAX N/A
EBX Frame Data Space
ECX N/A
EDX N/A
EBP Adapter Data Space
ESI ECB
EDI N/A
```

Note: Interrupts are in any state.

; On Return:

```
EAX Destroyed
EBX Preserved
ECX Destroyed
EDX Destroyed
EBP Preserved
ESI Preserved
EDI Preserved
```

Flags:

Note: Interrupts preserved.

; Remarks: This routine is called by AddAddress and OpenChannel.
; It is called at process time.

; See Also:

; END_MANUAL_ENTRY

public AddAddr
proc

```
mov eax, dword ptr [esi].CfgAddress
DebugMessage1 DEBUG_IOCTL, AddAddrMsg, eax
```

mov ecx, [esi].CfgNumAddresses ; ECX = Number 0

AddAddrLoop:

; First make sure this address is not a duplicate

```
lea edi, [ebp].Filter
mov edx, MAX_ADDR
AddAddrDuplicateLoop:
cmp [edi].FilterChannel, RBD_NOT_USED
je AddAddrDuplicateNext
mov eax, dword ptr [edi].FilterAddress
cmp eax, dword ptr [esi].CfgAddress
jne AddAddrDuplicateNext
mov ax, word ptr [edi].FilterAddress+4
cmp ax, word ptr [esi].CfgAddress+4
jne AddAddrDuplicateNext
```

mov eax, -1
ret

AddAddrDuplicateNext:

```
add edi, size FilterStruct
dec edx
jne AddAddrDuplicateLoop
```

; Find an empty slot in the filter table

```
lea edi, [ebp].Filter
xor edx, edx
AddAddrFindEmptyLoop:
cmp [edi].FilterChannel, RBD_NOT_USED
je AddAddrFindEmptyFound
add edi, size FilterStruct
inc edx
cmp edx, MAX_ADDR
jnb AddAddrFindEmptyLoop
```

mov eax, -1
ret

AddAddrFindEmptyFound:

```
mov eax, [esi].CfgChannel
mov [edi].FilterChannel, eax
mov eax, dword ptr [esi].CfgAddress
mov dword ptr [edi].FilterAddress, eax
mov ax, word ptr [esi].CfgAddress+4
mov word ptr [edi].FilterAddress+4, ax
mov [edi].FilterTotalCount, 0
mov [edi].FilterSeqCount, 0
mov [edi].FilterSeqNum, 0
```

```
mov edx, 16
mov eax, 31
test [esi].CfgAddress+0, 02h
jnz AddAddrFindEmptyLoop
mov edx, 2
mov eax, 13
```

AddAddrFindEblkLoop:

```

cmp     [ebp].EblkBusyFlags[edx], 0
je      AddAddrFindEblkFound
inc     edx
cmp     AddAddrFindEblkLoop
jbe     eax, -1
ret

```

AddAddrFindEblkFound:

```

mov     [ebp].EblkBusyFlags[edx], 1
mov     [edi].FilterCmdBlinkIndex, edx
mov     eax, [edi].FilterChannel
lea     edi, [ebp].Eblk
mov     [edi].EblkCmd, 0
mov     [edi].EblkPortID, ax
mov     ax, word ptr [esi].CfgAddress
xchg    ah, al
word ptr [edi].EblkAddress, ax
mov     ax, word ptr [esi].CfgAddress+2
xchg    ah, al
word ptr [edi].EblkAddress+2, ax

```

```

cmp     edx, 16
jae     AddAddrIBypass
mov     eax, dword ptr [esi].CfgGroupKey
mov     dword ptr [edi].EblkGroupKey, eax
mov     eax, dword ptr [esi].CfgGroupKey+4
mov     dword ptr [edi].EblkGroupKey+4, eax
mov     eax, dword ptr [esi].CfgElementKey
mov     dword ptr [edi].EblkElementKey, eax
mov     eax, dword ptr [esi].CfgElementKey+4
mov     dword ptr [edi].EblkElementKey+4, eax

```

AddAddrIBypass:

```

pushfd
cli
push    ecx
mov     eax, edx
mov     ecx, size EblkStruct
mul     ecx
mov     edx, [ebp].IOMsgRamPtr
add     eax, 18100h
shr     eax, 1
push    eax
push    esi
out     dx, ax
mov     edx, [ebp].IOMsgRam
mov     ecx, size EblkStruct / 2
mov     esi, edi
cld

```

AddAddrCopyEblk:

```

lodsw
out     dx, ax
dec     ecx
AddAddrCopyEblk
jne     AddAddrCopyEblk
pop     esi
pop     eax
pop     ecx
mov     edx, [ebp].IOMsgRamPtr
out     dx, ax
mov     edx, [ebp].IOMsgRam
mov     eax, 1

```

```

out     dx, ax
popfd

```

```

dec     ecx
jne     AddAddrLoop
xor     eax, eax
ret

```

AddAddr endp

```

subttl -- DeleteAddress --
page

```

```

;*****
; BEGIN_MANUAL_ENTRY( DeleteAddress, DPC/API/DELADDR )
;
; Name: DeleteAddress
; Description: This routine is called by DriverManagement to
;             delete the address passed in.
; On Entry:  EAX N/A
;            EBX Frame Data Space
;            ECX N/A
;            EDX N/A
;            EBP Adapter Data Space
;            ESI ECB
;            EDI N/A
; Note: Interrupts are in any state.
; On Return: EAX Destroyed
;            EBX Preserved
;            ECX Destroyed
;            EDX Destroyed
;            EBP Preserved
;            ESI Preserved
;            EDI Preserved
; Flags:
; Note: Interrupts preserved.
; Remarks: This routine is called by DriverManagement.
;           It is called at process time.
; See Also:
; END_MANUAL_ENTRY
;*****

```

```

public DeleteAddress
proc

```

```

mov     esi, [esi].RPacketOffset
mov     eax, [esi].CfgChannel
cmp     eax, MAX_CHAN
ja      DeleteAddressError
lea     edi, [ebp].RxControl[eax*8]
cmp     [edi].RxChannel, RBD_NOT_USED
je      DeleteAddressError
mov     ecx, [esi].CfgNumAddresses

```

```
cmp     ecx, MAX_CONF_ADDR
ja      DeleteAddressError
```

```
lea     ebx, [esi].CfgAddress
DeleteAddressLoop:
```

```
mov     ch, MAX_ADDR
lea     edi, [ebp].Filter
DeleteAddressFilterLoop:
mov     eax, dword ptr [ebx+0]
cmp     eax, dword ptr [edi].FilterAddress+0
jne     DeleteAddressNextFilter
mov     ax, word ptr [ebx+4]
cmp     ax, word ptr [edi].FilterAddress+4
jne     DeleteAddressNextFilter
```

```
mov     eax, [esi].CfgChannel
cmp     eax, [edi].FilterChannel
jne     DeleteAddressNextFilter
```

```
mov     [edi].FilterChannel, RBD_NOT_USED
mov     eax, [edi].FilterCmdBkIndex
mov     [ebp].EblkBusyFlags[eax], 0
```

```
pushfd
```

```
cld
```

```
push    ecx
mov     ecx, size EblkStruct
```

```
mul     ecx
mov     edx, [ebp].IOMsgRamPtr
add     eax, 18100h
```

```
shr     eax, 1
```

```
add     eax, 3
```

```
push    eax
```

```
out     dx, ax
```

```
mov     edx, [ebp].IOMsgRam
```

```
xor     eax, eax
```

```
out     dx, ax
```

```
out     dx, ax
```

```
pop     eax
```

```
pop     ecx
```

```
sub     eax, 3
```

```
mov     edx, [ebp].IOMsgRamPtr
```

```
out     dx, ax
```

```
mov     edx, [ebp].IOMsgRam
```

```
mov     eax, 1
```

```
out     dx, ax
```

```
out     dx, ax
```

```
popfd
```

```
DeleteAddressNextFilter:
```

```
add     edi, size FilterStruct
```

```
dec     ch
```

```
jne     DeleteAddressFilterLoop
```

```
DeleteAddressNext:
```

```
add     ebx, 6
```

```
dec     cl
```

```
jne     DeleteAddressLoop
```

```
xor     eax, eax
ret
```

```
DeleteAddressError:
```

```
mov     eax, -1
```

```
ret
```

```
DeleteAddress      endp
subttl             -- RegisterAgentSendRoutine --
page
```

```
*****
BEGIN_MANUAL_ENTRY( RegisterAgentSendRoutine, DPC/API/DELADDR )
*****
```

```
Name:              RegisterAgentSendRoutine
```

```
Description:       This routine is called by DriverManagement to
                    register a Slip Send routine. The first dword in the first
                    ECB fragment points to the Slip Send routine to use. To
                    deregister the send routine, set the dword to a NULL.
```

```
On Entry:
```

```
EAX    N/A
EBX    Frame Data Space
ECX    N/A
EDX    N/A
EBP    Adapter Data Space
ESI    ECB
EDI    N/A
```

```
Note:              Interrupts are in any state.
```

```
On Return:
```

```
EAX    Destroyed
EBX    Preserved
ECX    Destroyed
EDX    Destroyed
EBP    Preserved
ESI    Preserved
EDI    Preserved
```

```
Flags:
```

```
Note:              Interrupts preserved.
```

```
Remarks:          This routine is called by DriverManagement.
                    It is called at process time.
```

```
See Also:
```

```
END_MANUAL_ENTRY
```

```
*****
RegisterAgentSendRoutine proc
```

```
mov     esi, [esi].RPacketOffset
; ESI -> config structur
```

```
mov     eax, [esi]
; EAX -> Slip Send Routine Address
mov     [ebp].AgentSendRoutine, eax
xor     eax, eax
; Save it for later
ret
```

```
RegisterAgentSendRoutine      endp
subttl             -- RegisterAgent --
page
```

```
*****
BEGIN_MANUAL_ENTRY( RegisterAgent, DPC/API/REGAG )
*****
```

```
; Name: RegisterAgent
;
; Description: This routine is called by DriverManagement to
; register package delivery/internet agent. The first dword
; in the first ECB fragment points to the Remove routine
; that we must call before we are removed.
;
; On Entry: EAX N/A
; EBX Frame Data Space
; ECK N/A
; EDX N/A
; EBP Adapter Data Space
; ESI ECB
; EDI N/A
;
; Note: Interrupts are in any state.
;
; On Return: EAX Destroyed
; EBX Preserved
; ECK Destroyed
; EDX Destroyed
; EBP Preserved
; ESI Preserved
; EDI Preserved
;
; Flags:
; Note: Interrupts preserved.
;
; Remarks: This routine is called by DriverManagement.
; It is called at process time.
;
; See Also:
;
; END_MANUAL_ENTRY
;
; *****/
RegisterAgent proc
    mov esi, [esi].RPacketOffset ; ESI -> config structur
    mov eax, [esi]
    ; EAX -> Slip Send Routine Address
    mov [ebp].AgentRemoveRoutine, eax ; Save it for later
    xor eax, eax
    ret
RegisterAgent endp
    subttl -- ReturnTCBCompleteRoutine --
    page
; *****\
; BEGIN_MANUAL_ENTRY ( ReturnTCBCompleteRoutine, DPC/API/RETADS )
;
; Name: ReturnTCBCompleteRoutine
;
; Description: This routine is called by DriverManagement to
; return the TCB Complete routine. The first dword
; in the first ECB fragment points to a LONG which we will
; return with a pointer to the TCB complete routine.
;
; On Entry: EAX N/A
; EBX Frame Data Space
; ECK N/A
```

```
; EDX N/A
; EBP Adapter Data Space
; ESI ECB
; EDI N/A
;
; Note: Interrupts are in any state.
;
; On Return: EAX Destroyed
; EBX Preserved
; ECK Destroyed
; EDX Destroyed
; EBP Preserved
; ESI Preserved
; EDI Preserved
;
; Flags:
; Note: Interrupts preserved.
;
; Remarks: This routine is called by DriverManagement.
; It is called at process time.
;
; See Also:
;
; END_MANUAL_ENTRY
;
; *****/
if TIMESTAMP
    extrn GetHighResolutionTimer: near
    extrn HighResolutionTimer: dword
    public DPCTimestamp
DPCTimestamp proc
    CPush
    call GetHighResolutionTimer
    and eax, 00ffffffh
    mov edx, timestamp_index
    mov ecx, [esp + Parm0]
    shl ecx, 24
    or eax, ecx
    mov [edx], eax
    add edx, 4
    cmp edx, timestamp_end
    jae short DPCTimestampBegin
DPCTimestampExit:
    mov timestamp_index, edx
    mov dword ptr [edx], '$$$'
    CPop
    ret
DPCTimestampBegin:
    mov edx, timestamp_begin
    jmp DPCTimestampExit
DPCTimestamp endp
endif
    subttl -- DriverTCBComplete --
    page
DriverTCBComplete proc
    CPush
```

```
mov     esi, (esp + Param0) ; esi -> TCB
mov     ebp, OurAdapterDataSpace ; ebp -> Adapter Data Space
inc     [ebp].MSMtxFreeCount ; Add to send resources
call    EthersMFASTSendComplete ; Give it back
```

```
test    [ebp].MSMStatusFlags, SHUTDOWN ; Don't get next send
jne     GetNextSendExit ; if we're shutting down.
```

```
GetNextSendLoop:
test    [ebp].MSMStatusFlags, TXQUEUED ; Any ECB's waiting.
jnz     short GetNextSendExit ; Jump if there is.
```

GetNextSendExit:

```
CPOP
ret
```

```
GetNextSendExit:
call    EthersMGetNextSend ; Get next send if any.
jne     short GetNextSendExit ; Jump if no send
call    DriverSend ; Send it.
jmp     GetNextSendLoop ; See if we have any more
```

DriverTCBComplete endp

ReturnTCBCompleteRoutine proc

```
mov     esi, [esi].RPacketOffset ; esi -> fragment
mov     dword ptr [esi], offset DriverTCBComplete
xor     eax, eax
ret
```

ReturnTCBCompleteRoutine endp
subttl -- OpenChannel --
page

```
*****\
; BEGIN_MANUAL_ENTRY ( OpenChannel, DPC/API/OPENCHAN )
```

Name: OpenChannel

Description: This routine is called by DriverManagement to
open a channel on the adapter to receive packets
from.

On Entry: EAX N/A
 EBX Frame Data Space
 ECX N/A

 EDX N/A
 EBP Adapter Data Space
 ESI ECB
 EDI N/A

Note: Interrupts are in any state.

On Return: EAX Destroyed
 EBX Preserved
 ECX Destroyed
 EDX Destroyed
 EBP Preserved
 ESI Preserved
 EDI Preserved

Flags:

```
Note:      Interrupts preserved.
Remarks:   This routine is called by DriverManagement.
            It is called at process time.
```

See Also:

END_MANUAL_ENTRY

public OpenChannel

```
OpenChannel
proc
mov     esi, [esi].RPacketOffset ; esi -> config structure
```

Don't open if we don't have signal lock.

```
cmp     [ebp].SignalQuality, 200
jb      OpenChannelError
```

Find an empty RxControl entry

```
xor     ecx, ecx ; ECX = index
lea     edi, [ebp].RxControl ; EDI -> Rx Control Structures
```

```
FindEmptyRBDLoop:
cmp     [edi].RxChannel, RBD_NOT_USED ; empty?
jne     FindEmptyRBDNext ; Jump if not
```

Found one. Initialize it before adding addresses.

```
mov     [edi].RxChannel, ecx ; Save channel number
mov     [esi].CfGChannel, ecx
mov     eax, [esi].CfGESR
mov     [edi].RxESR, eax
```

```
push    edi
call    AddAddr
pop     edi
or      eax, eax
jne     OpenChannelDidntAdd
ret
```

OpenChannelDidntAdd:

```
mov     [edi].RxChannel, RBD_NOT_USED
mov     [edi].RxESR, 0
mov     eax, -1
ret
```

FindEmptyRBDNext:

```
inc     ecx
add     edi, size RX_CNTL
cmp     ecx, MAX_CHAN
jb      FindEmptyRBDLoop
```

OpenChannelError:

```
mov     eax, -1
ret
```

OpenChannel endp
subttl -- RefreshMipsStats --
page

*****\
; BEGIN_MANUAL_ENTRY (RefreshMipsStats, DPC/API/RFSHMIPS)

RefreshMipsStats

Name: RefreshMipsStats

Description: This routine is called by to read the Mips stats from the adapter and to store them into the Local Mips Stats structure.

On Entry:

EAX N/A

EBX N/A

ECX N/A

EDX N/A

EBP Adapter Data Space

ESI N/A

EDI N/A

Note: Interrupts are in any state.

On Return:

EAX Destroyed

EBX Preserved

ECX Destroyed

EDX Destroyed

EBP Preserved

ESI Preserved

EDI Destroyed

Flags:

Note: Interrupts preserved.

Remarks: This routine is called by GetMipsStats and DriverCallback. It can be called at process or interrupt time.

See Also:

END_MANUAL_ENTRY

RefreshMipsStats proc

```
pushfd
cli
mov     edx, [ebp].IOMsgRamPcr
mov     eax, 18700h / 2
out     dx, ax

mov     edx, [ebp].IOMsgRam
lea     edi, [ebp].MipsRxEnables
mov     ecx, (size StatsBk) / 2

cld

GetMipsStatsLoop:
in      ax, dx
stosw
loop   GetMipsStatsLoop

popfd
xor     eax, eax
ret
```

RefreshMipsStats endp

subttl -- GetMipsStats --

page

BEGIN_MANUAL_ENTRY (GetMipsStats, DPC/API/GETMIPS)

GetMipsStats

Name: GetMipsStats

Description: This routine is called by DriverManagement to return the Mips statistics.

On Entry:

EAX N/A

EBX Frame Data Space

ECX N/A

EDX N/A

EBP Adapter Data Space

ESI ECB

EDI N/A

Note: Interrupts are in any state.

On Return:

EAX Destroyed

EBX Preserved

ECX Destroyed

EDX Destroyed

EBP Preserved

ESI Preserved

EDI Preserved

Flags:

Note: Interrupts preserved.

Remarks: This routine is called by DriverManagement. It is called at process time.

See Also:

END_MANUAL_ENTRY

GetMipsStats proc

```
push     esi
call     RefreshMipsStats
pop      esi

mov     edi, [esi].RPacketOffset
lea     esi, [ebp].MipsRxEnables
mov     ecx, (size StatsBk) / 4

cld

rep     movsd

xor     eax, eax
ret
```

GetMipsStats endp

.....

BEGIN_MANUAL_ENTRY (DriverMulticastChange, DPC/API/MULTI)

Name: DriverMulticastChange

Description: This routine will modify the NIC's multicast registers to enable it to receive the multicast addresses listed in the multicast table. Each entry in the multicast table is as follows:

bytes 0-5 = Multicast Address.
bytes 6-7 = Entry used(Non zero if used).

On Entry:

EAX N/A
EBX N/A
ECX # of Entries in Table(0 if empty)
EDX N/A
EBP @ Adapter Data Space
ESI @ Multicast Table
EDI N/A

Note: Interrupts are in any state.

On Return:

EAX Destroyed
EBX Preserved
ECX Destroyed
EDX Destroyed
EBP Preserved
ESI Destroyed
EDI Destroyed

Flags:

Note: Interrupts preserved.

Remarks:

This routine is called by the ethernet media module.
It can be called at process or interrupt time.

See Also:

ETHERTSM\ethertsmaddmulticastAddress
ETHERTSM\ethertsmdeletemulticastAddress
ETHERTSM\ethertsmupdateMulticast

END_MANUAL_ENTRY

DriverMulticastChange proc

First reset Multicast Address Registers.

ret

DriverMulticastChange endp

subttl -- DriverPromiscuousChange --
page

BEGIN_MANUAL_ENTRY(DriverPromiscuousChange, DPC/API/PROMISCU)

Name: DriverPromiscuousChange

Description: This routine will enable/disable the Promiscuous Mode.

On Entry:

EAX N/A
EBX N/A
ECX 0 to disable the Promiscuous mode
EDX N/A
EBP @ Adapter Data Space
ESI @ Multicast Table
EDI N/A

Note: Interrupts are in any state.

On Return:

EAX Destroyed
EBX Preserved
ECX Destroyed
EDX Destroyed
EBP Preserved
ESI Destroyed
EDI Destroyed

Flags:

Note: Interrupts preserved.

Remarks:

This routine is called by the ethernet media module.
It can be called at process or interrupt time.

See Also: ETHERTSM\ethertsmPromiscuousChange

END_MANUAL_ENTRY

DriverPromiscuousChange proc

ret

DriverPromiscuousChange endp

subttl -- CalculatedDriftDelta --
page

BEGIN_MANUAL_ENTRY(CalculatedDriftDelta, DPC/API/CALCDD)

Name: CalculatedDriftDelta

Description: Acquisition State Routine.

On Entry:

EAX N/A
EBX Frame Data Space
ECX N/A
EDX N/A
EBP Adapter Data Space
ESI N/A
EDI N/A

Note: Interrupts are in any state.

On Return:

EAX Destroyed
EBX Preserved
ECX Destroyed
EDX Destroyed
EBP Preserved
ESI Preserved
EDI Preserved

Flags:

Note: Interrupts preserved.

Remarks:

This routine is called by InitState.
It can be called at process or interrupt time.

See Also:

END_MANUAL_ENTRY

public CalculatedDriftDelta
proc

```
mov     edi, [ebp].Drift  
cmp     edi, NOM_COUNT_TRACK  
jbe     DriftBelowNOM  
  
lea     eax, [edi - NOM_COUNT_TRACK]  
xor     edx, edx  
mov     ecx, 210  
div     ecx  
mov     [ebp].GLDrift, eax  
  
mov     ecx, 210  
mul     ecx  
shr     eax, 4  
mov     edi, eax  
mov     eax, [ebp].Drift  
sub     eax, NOM_COUNT_TRACK  
sub     eax, edi  
  
mov     edi, 0ffffh  
sub     edi, [ebp].GLDrift  
inc     edi  
mov     [ebp].GLDrift, edi  
  
ret
```

DriftBelowNOM:

```
mov     eax, NOM_COUNT_TRACK  
sub     eax, [ebp].Drift  
xor     edx, edx  
mov     ecx, 210  
div     ecx  
mov     [ebp].GLDrift, eax  
  
mov     ecx, 210  
mul     ecx  
shr     eax, 4  
mov     edi, NOM_COUNT_TRACK  
sub     edi, [ebp].Drift  
sub     edi, eax  
  
mov     eax, 0ffffh  
sub     eax, edi  
inc     eax  
ret
```

CalculatedDriftDelta endp
subttl -- Step --
page

BEGIN_MANUAL_ENTRY(Step, DPC/API/STEP)

Name: / Step

Description: Acquisition State Routine.

On Entry: EAX N/A

EBX	Frame Data Space
ECX	N/A
EDX	N/A
EBP	Adapter Data Space
ESI	N/A
EDI	N/A

Note: Interrupts are in any state.

On Return:	EAX	Destroyed
	EBX	Preserved
	ECX	Destroyed
	EDX	Destroyed
	EBP	Preserved
	ESI	Preserved
	EDI	Preserved

Flags:

Note: Interrupts preserved.

Remarks: This routine is called by InitState.
It can be called at process or interrupt time.

See Also:

END_MANUAL_ENTRY

Step proc

```
mov     eax, [ebp].NextStepCount  
inc     eax  
xor     edx, edx  
mov     ecx, 4  
div     ecx  
mov     [ebp].NextStepCount, edx  
  
mov     eax, [ebp].SearchLoc  
cmp     [ebp].SearchLocFound, FALSE  
je
```

```
or     edx, edx  
StepSetGLOffset  
cmp     edx, 2  
je     StepSetGLOffset
```

```
inc     eax  
cmp     edx, 1  
je     StepDivideBy3  
inc     eax  
StepDivideBy3:
```

```
xor     edx, edx  
mov     ecx, 3  
div     ecx  
mov     eax, edx
```

StepSetGLOffset:

```
mov     eax, SearchTbl[eax * 4]  
mov     [ebp].GLOffset, eax  
jmp     StepCalcRx
```

DontUseNextStep:

```
mov     ecx, SearchTbl[eax * 4]
```

```
mov     [ebp].GLOffset, ecx
inc     eax
xor     edx, edx
mov     ecx, 3
div     ecx
mov     [ebp].GLOffset, edx
```

StepCalcRx:

```
mov     [ebp].Drift, 0
call    CalculateRxFreq
mov     [ebp].Drift, NOM_COUNT_TRACK
ret
```

```
Step    endp
subttl  -- InitState --
page
```

```
.....
; BEGIN_MANUAL_ENTRY( InitState, DPC/API/INITSTA )
```

```
; Name: InitState
```

```
; Description: Acquisition State Routine.
```

```
; On Entry: EAX N/A
;           EBX Frame Data Space
;           ECX N/A
;           EDX N/A
;           EBP Adapter Data Space
;           ESI N/A
;           EDI N/A
```

```
Note: Interrupts are in any state.
```

```
On Return: EAX Destroyed
;           EBX Preserved
;           ECX Destroyed
;           EDX Destroyed
;           EBP Preserved
;           ESI Preserved
;           EDI Preserved
```

```
Flags:
```

```
Note: Interrupts preserved.
```

```
Remarks: This routine is called by DriverCallback.
;         It can be called at process or interrupt time.
```

```
; See Also:
```

```
; END_MANUAL_ENTRY
```

```
.....
; public InitState
```

```
InitState proc
```

```
cmp     [ebp].TrackingMode, TRUE
jne     InitStateNotTracking
cmp     DbgMsgMark, 0
je      InitStateNoMsg
```

```
mov     eax, offset InitStateTrackMsg
cmp     eax, LastDebugMessage
je      InitStateNoMsg
push    LastDebugMessage, eax
push    eax
push    DPCScreen
call    OutputToScreen
lea     esp, [esp + (2 * 4)]
```

```
InitStateNoMsg:
```

```
call    CalculateIfDelta
add     eax, NOM_COUNT_REACQ
```

```
mov     edx, [ebp].IOCountNomLowAddr
out     dx, al
shr     al, 8
mov     edx, [ebp].IOCountNomHighAddr
out     dx, al
```

```
mov     edx, [ebp].IOGateCountHighAddr
mov     eax, [ebp].ReacqGateCount
out     dx, al
```

```
mov     edx, [ebp].IOSyntheserControlAddr
in      al, dx
or      al, RESET_FEC_ACQ_MASK
out     dx, al
```

```
mov     edx, [ebp].IOBtrControlAddr
xor     eax, eax
out     dx, ax
```

```
mov     edx, [ebp].IOAfControlAddr
in      al, dx
or      al, SWP_ENA_MASK
out     dx, al
```

```
mov     edx, [ebp].IOBtrControlAddr
in      al, dx
or      al, FREQ_PWR_OFFSET OR 6 OR BTR_SENSE_MASK
out     dx, al
```

```
mov     edx, [ebp].IOSyntheserControlAddr
in      al, dx
or      al, RESET_FEC_ACQ_MASK
out     dx, al
```

```
mov     edx, [ebp].IOSyntheserControlAddr
in      al, dx
cmp     [ebp].ViterbiMode, LOWRATE
jne     InitStateSetMode
and     al, NOT MODE_MASK
jmp     InitStateCheckRate
```

```
InitStateSetMode:
```

```
or      al, MODE_MASK
```

```
InitStateCheckRate:
out     dx, al
```

```
mov     edx, [ebp].IOSpareIOControlAddr
mov     al, 0ah
cmp     [ebp].ViterbiOnly, 2
je      InitStateSetRate
mov     al, 0bh
cmp     [ebp].ViterbiOnly, 1
je      InitStateSetRate
```

```

mov     al, 0fh
InitStateSetRate:
out     dx, al
mov     [ebp].NextState, ENABLE_BTR
jmp     InitStateCheckPointing

```

InitStateNotTracking:

```

cmp     DebugMask, 0
je      InitStateNOMsg
mov     eax, offset InitStateNotTrackMsg
cmp     eax, LastDebugMessage
je      InitStateNOMsg
mov     LastDebugMessage, eax
push    eax
push    DPCScreen
call    OutputToScreen
lea     esp, [esp + (2 * 4)]
InitStateNOMsg:

```

```

mov     edx, [ebp].IOBitDetControlAddr
xor     eax, eax
out     dx, al
mov     edx, [ebp].IOSpareIOControlAddr
mov     al, 0ah
mov     [ebp].ViterbiOnly, 2
cmp     InitStateTSetRate
je      al, 0bh
mov     [ebp].ViterbiOnly, 1
cmp     InitStateTSetRate
je      al, 0fh
mov     InitStateTSetRate:
out     dx, al

```

```

mov     edx, [ebp].IODAOffsetControlAddr
xor     eax, eax
out     dx, al
mov     edx, [ebp].IOAfControlAddr
mov     eax, [ebp].ModulationScheme
or      eax, SWEET_DIR_SENSE_MASK
out     dx, al

```

```

mov     edx, [ebp].IOSweepRateAddr
mov     al, 8ah
out     dx, al
mov     cr1x, [ebp].IOCR1ControlHighAddr
mov     eax, [ebp].ReacquiredCount
out     dx, al

```

```

mov     edx, [ebp].IOCountDeltaAddr
mov     eax, [ebp].SgfdeltaCount
out     dx, al
mov     eax, [ebp].NomCountSearch
mov     [ebp].TuneCount, eax
mov     edx, [ebp].IOCountNomLowAddr
out     dx, al
shr     eax, 8
mov     edx, [ebp].IOCountNomHighAddr
out     dx, al

```

```

mov     edx, [ebp].IOSynthesizerControlAddr
in      al, dx

```

```

or      al, RESET_FEC_ACQ_MASK
out     dx, al
mov     edx, [ebp].IOBtrControlAddr
xor     eax, eax
out     dx, al

```

```

mov     edx, [ebp].IOAfControlAddr
in      al, dx
or      al, SWP_ENA_MASK
out     dx, al
mov     edx, [ebp].IOAgcFirControlAddr
mov     al, 10 OR AGC_SENSE_MASK
out     dx, al

```

```

mov     edx, [ebp].IOBtrControlAddr
in      al, dx
or      al, FREQ_PWR_OFFSET OR 6 OR BTR_SENSE_MASK
out     dx, al

```

```

mov     edx, [ebp].IOCtrlLowAddr
mov     al, 60h
out     dx, al
mov     edx, [ebp].IOCtrlAddr
mov     al, 0e0h
out     dx, al

```

```

mov     edx, [ebp].IOSynthesizerControlAddr
mov     al, SENA_MASK
cmp     [ebp].ModulationScheme, BPSK
jne     InitStateSetSena
or      al, DEPUNC_BYPASS_MASK
InitStateSetSena:
out     dx, al

```

```

mov     edx, [ebp].IOCtrlControlAddr
mov     al, 16 OR CRUK_GAIN_OFFSET OR CRUK_DET_PWR_OFFSET
out     dx, al
mov     edx, [ebp].IOSynthesizerControlAddr
in      al, dx
cmp     [ebp].ViterbiMode, LOWRATE
jne     InitStateResetBtr

```

```

or      al, RESET_BTR_ACC_MASK
and     al, NOT MODE_MASK
jmp     InitStateResetBtr

```

```

InitStateResetBtr:
or      al, MODE_MASK OR RESET_BTR_ACC_MASK
InitStateClearBtr:
out     dx, al

```

```

in      al, dx
and     al, NOT RESET_BTR_ACC_MASK
out     dx, al
call    Step

```

```

mov     [ebp].NextState, ACQ_PD
InitStateCheckPointing:

```

```

mov     [ebp].RateCount, 0
mov     [ebp].PointingFlag, TRUE

```

```
cmp      [ebp].DemoCommand, POINTING_MODE
je       InitStateExit
mov      [ebp].PointingFlag, FALSE
InitStateExit:
mov      mov      [ebp].CurrentState, SIXTH_PRGM
mov      [ebp].SignalQuality, 0
mov      [ebp].DemoCommand, BUSY_MODE
mov      [ebp].DemoStatus, UNLOCKED
mov      [ebp].FecStatus, UNLOCKED
ret

InitState
subttl   -- ProgTuner --
page

;.....\

; BEGIN_MANUAL_ENTRY ( ProgTuner, DPC/API/PROGTUN )
;
; Name:      ProgTuner
;
; Description: Acquisition State Routine.
;
; On Entry:
;   EAX      N/A
;   EBX      Frame Data Space
;   ECX      N/A
;   EDX      N/A
;   EBP      Adapter Data Space
;   ESI      N/A
;   EDI      N/A
;
; Note:      Interrupts are in any state.
;
; On Return:
;   EAX      Destroyed
;   EBX      Preserved
;   ECX      Destroyed
;   EDX      Destroyed
;   EBP      Preserved
;   ESI      Preserved
;   EDI      Preserved
;
; Flags:
;
; Note:      Interrupts preserved.
;
; Remarks:   This routine is called by Tune.
;            It can be called at process or interrupt time.
;
; See Also:
;
; END_MANUAL_ENTRY
;...../

public   ProgTuner
ProgTuner
proc
;
; EAX = data
; ECX = len
;
;   dec      ecx
;   mov      edx, 1
;   shl      edx, cl
;   mov      ecx, edx
;   mov      esi, eax
;
;   ; ESI = Data
```

```
ProgTunerLoop:
jecxz    ProgTunerExit
mov      mov      [ebp].IOSynthesizerControlAddr
in      al, dx
test     esi, ecx
je       ProgTunerClear
or       al, SDATA_MASK
and      al, NOT SCLK_MASK
out      dx, al
jmp      ProgTunerDelay

ProgTunerClear:
and      al, NOT (SCLK_MASK OR SDATA_MASK)
out      dx, al

ProgTunerDelay:
shr      ecx, 1
mov      mov      [ebp].IOStatusAddr
in      al, dx
in      al, dx
or       al, SCLK_MASK
out      dx, al

mov      mov      [ebp].IOSynthesizerControlAddr
in      al, dx
in      al, SCLK_MASK
out      dx, al

mov      mov      [ebp].IOStatusAddr
in      al, dx
in      al, dx

jmp      ProgTunerLoop

ProgTunerExit:
mov      mov      [ebp].IOSynthesizerControlAddr
in      al, dx
and      al, NOT SCLK_MASK
out      dx, al

ret

ProgTuner
subttl   -- Tune --
page

;.....\

; BEGIN_MANUAL_ENTRY ( Tune, DPC/API/TUNE )
;
; Name:      Tune
;
; Description: Acquisition State Routine.
;
; On Entry:
;   EAX      N/A
;   EBX      Frame Data Space
;   ECX      N/A
;   EDX      N/A
;   EBP      Adapter Data Space
;   ESI      N/A
;   EDI      N/A
;
; Note:      Interrupts are in any state.
;
; On Return:
;   EAX      Destroyed
```

EBX Preserved
ECX Destroyed
EDX Destroyed
EBP Preserved
ESI Preserved
EDI Preserved

Flags:

Note: Interrupts preserved.

Remarks: This routine is called by SynthPrgmState.
It can be called at process or interrupt time.

See Also:

END_MANUAL_ENTRY

...../

Tune proc

cmp [ebp].TunerTypeFound, SHARP
je TuneSharpPan
cmp [ebp].TunerTypeFound, PANASONIC
je TuneSharpPan
cmp [ebp].TunerTypeFound, SHARP_CUSTOM
je TuneSharpCustom
ret

TuneSharpPan:

mov edx, [ebp].IOSynthSerControlAddr
in al, dx
or al, SENA_MASK
out dx, al

mov edx, [ebp].IOSynthSerControlAddr
in al, dx
and al, NOT (SCLK_MASK OR SDATA_MASK)
out dx, al

cmp [ebp].TrackingMode, 0
jne TuneSetNA

mov edx, [ebp].IOSynthSerControlAddr
in al, dx
and al, NOT SENA_MASK
out dx, al

mov eax, 2ch
cmp [ebp].TunerTypeFound, SHARP
je TuneProgTuner
mov eax, 0ech

TuneProgTuner:

mov ecx, 8
call ProgTuner

mov edx, [ebp].IOSynthSerControlAddr
in al, dx
or al, SENA_MASK
out dx, al

mov edx, [ebp].IOSynthSerControlAddr
in al, dx
in al, dx

mov edx, [ebp].IOSynthSerControlAddr
in al, dx
and al, NOT SENA_MASK
out dx, al

mov eax, 28h OR 2000h
mov ecx, 16
call ProgTuner

mov edx, [ebp].IOSynthSerControlAddr
in al, dx
or al, SENA_MASK
out dx, al

mov edx, [ebp].IOSynthSerControlAddr
in al, dx
in al, dx

TuneSetNA:

mov edx, [ebp].IOSynthSerControlAddr
in al, dx
and al, NOT SENA_MASK
out dx, al

mov eax, [ebp].ChannelNumber
add eax, [ebp].GLDrift
xor edx, edx
mov ecx, SYNTH_RATIO
div ecx
mov edi, edx
or eax, 3000h

mov ecx, 16
call ProgTuner

mov eax, edi
mov ecx, 8
call ProgTuner

mov edx, [ebp].IOSynthSerControlAddr
in al, dx
or al, SENA_MASK
out dx, al

ret

TuneSharpCustom:

mov edx, [ebp].IOSynthSerControlAddr
in al, dx
and al, NOT SENA_MASK
out dx, al

mov edx, [ebp].IOSynthSerControlAddr
in al, dx
and al, NOT (SCLK_MASK OR SDATA_MASK)
out dx, al

mov eax, 50h OR 8001h
mov ecx, 16
call ProgTuner

mov edx, [ebp].IOSynthSerControlAddr
in al, dx
or al, SENA_MASK
out dx, al

```

mov     edx, [ebp].IOStatusAddr
in      al, dx
in      al, dx
mov     edx, [ebp].IOSyntheserControlAddr
in      al, dx
and     al, NOT SENA_MASK
out     dx, al

mov     eax, [ebp].ChannelNumber
add     eax, [ebp].GLDOffset
xor     edx, edx
mov     ecx, SYNTH_RATIO
div     ecx, edx
mov     edi, edx

mov     ecx, 11
call    ProgTuner

mov     eax, edi
shl     eax, 1
mov     ecx, 9
call    ProgTuner

mov     edx, [ebp].IOSyntheserControlAddr
in      al, dx
or      al, SENA_MASK
out     dx, al

mov     edx, [ebp].IOStatusAddr
in      al, dx
in      al, dx
mov     edx, [ebp].IOSyntheserControlAddr
in      al, dx
and     al, NOT SENA_MASK
out     dx, al

ret

Tune    endp
subttl  -- SynchPrgmState --
page
;.....\
;
; BEGIN_MANUAL_ENTRY( SynchPrgmState, DPC/API/SYNTHPS )
;
; Name:      SynchPrgmState
;
; Description: Acquisition State Routine.
;
; On Entry:
;
;   EAX      N/A
;   EBX      Frame Data Space
;   ECX      N/A
;   EDX      N/A
;   EBP      Adapter Data Space
;   ESI      N/A
;   EDI      N/A
;
; Note:      Interrupts are in any state.
;
; On Return:
;   EAX      Destroyed
;   EBX      Frame Data Space
;   ECX      Destroyed

```

```

;
;   EDX      Destroyed
;   EBP      Preserved
;   ESI      Preserved
;   EDI      Preserved
;
; Flags:
;
; Note:      Interrupts preserved.
;
; Remarks:
;   This routine is called by DriverCallBack.
;   It can be called at process or interrupt time.
;
; See Also:
;
; END_MANUAL_ENTRY
;
;...../
;
; public SynchPrgmState
; SynchPrgmState proc
;
;   cmp     DebugMask, 0
;   je      SynchPrgmStateNoMsg
;   mov     eax, offset SynchPrgmMsg
;   cmp     eax, lastDebugMessage
;   je      SynchPrgmStateNoMsg
;   mov     eax, lastDebugMessage
;   push    eax
;   push    DPCScreen
;   call    OutputToScreen
;   lea     esp, [esp + (2 * 4)]
;   SynchPrgmStateNoMsg:
;   call    Tune
;
;   mov     [ebp].TrackingMode, 0
;   cmp     [ebp].NextState, ACQ_PD
;   jne     SynchPrgmClearT2
;
;   mov     [ebp].MaxSgf, 0
;   mov     [ebp].SgfAvg, 0
;   mov     [ebp].SgfWait, 0
;   mov     eax, [ebp].SgfCheckPoints
;   mov     [ebp].MaxCount, eax
;   mov     [ebp].T2Count, 60
;   mov     [ebp].CurrentState, ACQ_PD_DELAY
;   ret
;
; SynchPrgmClearT2:
;   mov     [ebp].T2Count, 0
;   mov     [ebp].CurrentState, ACQ_PD_DELAY
;   ret
;
; SynchPrgmState endp
; subttl  -- AcqPDDelayState --
; page
;.....\
;
; BEGIN_MANUAL_ENTRY( AcqPDDelayState, DPC/API/ACQPDPS )
;
; Name:      AcqPDDelayState
;
; Description: Acquisition State Routine.
;
; On Entry:
;   EAX      N/A
;   EBX      Frame Data Space

```

ECX N/A
 EDX N/A
 EBP Adapter Data Space
 ESI N/A
 EDI N/A

Note: Interrupts are in any state.

On Return:

EAX Destroyed
 EBX Preserved
 ECX Destroyed
 EDX Destroyed
 EBP Preserved
 ESI Preserved
 EDI Preserved

Flags:

Note: Interrupts preserved.

Remarks:

This routine is called by DriverCallback.
 It can be called at process or interrupt time.

See Also:

END_MANUAL_ENTRY

```
public AcqPDDelayState
AcqPDDelayState proc
```

```
    cmp     DebugMask, 0
    je      AcqPDDelayStateNomsg
    mov     eax, offset AcqPDDelayMsg
    cmp     eax, LastDebugMessage
    je      AcqPDDelayStateNomsg
    mov     eax, LastDebugMessage
    push    eax
    push    DPCScreen
    call    OutputToScreen
    lea     esp, [esp + (2 * 4)]
    AcqPDDelayStateNomsg:
    cmp     [ebp].T2Count, 0
    jne     AcqPDDelayExit
```

```
    mov     edx, [ebp].IOSweepRateAddr
    mov     al, 87h
    out     dx, al

    mov     edx, [ebp].IOAfcControlAddr
    in      al, dx
    and     al, NOT SQF_PEAK_EN_MASK
    out     dx, al

    mov     eax, [ebp].SqfWait
    [ebp].T2Count, eax

    mov     eax, [ebp].NextState
    [ebp].CurrentState, eax

    mov     edx, [ebp].IOAfcControlAddr
    in      al, dx
    or      al, SQF_PEAK_EN_MASK
    out     dx, al
```

AcqPDDelayExit:

ret

```
AcqPDDelayState endp
subttl -- AcqPDDState --
page
```

```
BEGIN_MANUAL_ENTRY( AcqPDDState, DPC/API/ACQPDS )
```

Name: AcqPDDState

Description: Acquisition State Routine.

On Entry:

EAX N/A
 EBX Frame Data Space
 ECX N/A
 EDX N/A
 EBP Adapter Data Space
 ESI N/A
 EDI N/A

Note: Interrupts are in any state.

On Return:

EAX Destroyed
 EBX Preserved
 ECX Destroyed
 EDX Destroyed
 EBP Preserved
 ESI Preserved
 EDI Preserved

Flags:

Note: Interrupts preserved.

Remarks:

This routine is called by DriverCallback.
 It can be called at process or interrupt time.

See Also:

END_MANUAL_ENTRY

```
public AcqPDDState
AcqPDDState proc
```

```
    cmp     DebugMask, 0
    je      AcqPDDStateNomsg
    mov     eax, offset AcqPDDMsg
    cmp     eax, LastDebugMessage
    je      AcqPDDStateNomsg
    mov     eax, LastDebugMessage
    push    eax
    push    DPCScreen
    call    OutputToScreen
    lea     esp, [esp + (2 * 4)]
    AcqPDDStateNomsg:
```

```
    cmp     [ebp].T2Count, 0
    jne     AcqPDDExit

    xor     eax, eax
    mov     edx, [ebp].IOAfcControlAddr
    in      al, dx
```

```

add     [ebp].SqfAvg, eax
cmp     eax, [ebp].MaxSqf
jbe     AcqPDDecMaxCount
mov     [ebp].MaxSqf, eax
mov     eax, [ebp].TuneCount
mov     [ebp].BestTuneCount, eax

```

AcqPDDecMaxCount:

```

dec     [ebp].MaxCount
jne     AcqPDMaxCountNotZero
mov     edx, [ebp].IOSweepRateAddr
mov     al, 8ah
out     dx, al

```

```

mov     edx, [ebp].IOCountNomLowAddr
mov     eax, [ebp].BestTuneCount
out     dx, al
shr     eax, 8
mov     edx, [ebp].IOCountNomHighAddr
out     dx, al

```

```

mov     edx, [ebp].IOCountDeltaAddr
mov     eax, [ebp].SqfDeltaCount
shl     eax, 1
out     dx, al

```

```

mov     eax, [ebp].SqfAvg
mov     ecx, [ebp].SqfCheckPoints
xor     edx, edx
div     ecx
add     eax, 2
mov     [ebp].SqfAvg, eax

```

```

mov     [ebp].T2Count, 40
mov     [ebp].NextState, ENABLE_BTR
mov     [ebp].CurrentState, ACQ_PD_DELAY

```

AcqPDExit:

ret

AcqPDMaxCountNotZero:

```

mov     eax, [ebp].TuneCount
add     add, [ebp].SqfCheckStepsSize
mov     [ebp].TuneCount, eax
mov     edx, [ebp].IOCountNomLowAddr
out     dx, al
mov     edx, [ebp].IOCountNomHighAddr
shr     eax, 8
out     dx, al

```

```

mov     [ebp].T2Count, 20
mov     [ebp].CurrentState, ACQ_PD_DELAY
ret

```

```

AcqPDState     endp
subttl        -- EnableBTRState --
page

```

BEGIN_MANUAL_ENTRY(EnableBTRState, DPC/API/ENDTRST)

Name: EnableBTRState

Description: Acquisition State Routine.

On Entry: EAX N/A

EBX Frame Data Space

ECX N/A

EDX N/A

EBP Adapter Data Space

ESI N/A

EDI N/A

Note: Interrupts are in any state.

On Return:

EAX Destroyed

EBX Preserved

ECX Destroyed

EDX Destroyed

EBP Preserved

ESI Preserved

Flags:

Note: Interrupts preserved.

Remarks: This routine is called by DriverCallback.

It can be called at process or interrupt time.

See Also:

END_MANUAL_ENTRY

```

public EnableBTRState
EnableBTRState
proc

```

```

    cmp     DebugMask, 0
    je      EnableBTRStateNomsg
    mov     eax, offset EnableBTRMsg
    cmp     eax, lastDebugMessage
    je      EnableBTRStateNomsg
    mov     LastDebugMessage, eax
    push    eax
    push    DPCScreen
    call    OutputToScreen
    lea     esp, [esp + (2 * 4)]
EnableBTRStateNomsg:
    mov     edx, [ebp].IOBtrControlAddr
    in      al, dx
    or      al, BTR_ERR_ENA_MASK
    out     dx, al

```

```

mov     [ebp].CurrentState, START_SEARCH_FOR_FEC
ret

```

```

EnableBTRState     endp
subttl        -- StartSearchForFECState --
page

```

BEGIN_MANUAL_ENTRY(StartSearchForFECState, DPC/API/SEARCH_FEC)


```
Name: StartSearchForFECState
Description: Acquisition State Routine.

On Entry:
    EAX N/A
    EBX Frame Data Space
    ECX N/A
    EDX N/A
    EBP Adapter Data Space
    ESI N/A
    EDI N/A

Note: Interrupts are in any state.

On Return:
    EAX Destroyed
    EBX Preserved
    ECX Destroyed
    EDX Destroyed
    EBP Preserved
    ESI Preserved
    EDI Preserved

Flags:
Note: Interrupts preserved.

Remarks: This routine is called by DriverCallBack.
It can be called at process or interrupt time.

See Also:

END_MANUAL_ENTRY
.....
public StartSearchForFECState
StartSearchForFECState proc

    cmp     DebugMask, 0
    je      StartSearchForFECStateNomsg
    mov     eax, offset StartSearchForFECMsg
    cmp     eax, LastDebugMessage
    je      StartSearchForFECStateNomsg
    mov     eax, LastDebugMessage
    push    eax
    push    DPCScreen
    call    OutputToScreen
    lea     esp, [esp + (2 * 4)]
    StartSearchForFECStateNomsg:
    cmp     [ebp].PointingFlag, 0
    je      SearchFECNotPointing

    mov     [ebp].CurrentState, POINTING_ACQ
    mov     eax, [ebp].SqfAvg
    add     eax, 2
    mov     [ebp].MaxSqf, eax
    jmp     SearchFECSetMax

SearchFECNotPointing:

    mov     [ebp].CurrentState, CHECK_FOR_FEC_LOCK
    mov     eax, [ebp].SqfAvg
    add     eax, 6
    mov     [ebp].MaxSqf, eax

SearchFECSetMax:
.....
```

```
Name: CheckforFEClockState
Description: Acquisition State Routine.

On Entry:
    EAX N/A
    EBX Frame Data Space
    ECX N/A
    EDX N/A
    EBP Adapter Data Space
    ESI N/A
    EDI N/A

Note: Interrupts are in any state.

On Return:
    EAX Destroyed
    EBX Preserved
    ECX Destroyed
    EDX Destroyed
    EBP Preserved
    ESI Preserved
    EDI Preserved

Flags:
Note: Interrupts preserved.

Remarks: This routine is called by DriverCallBack.
It can be called at process or interrupt time.

See Also:

END_MANUAL_ENTRY
.....
public CheckforFEClockState
.....
```

CheckForFEClockState proc

```
    cmp     DebugMask, 0
    je      CheckForFEClockStateNomsg
    mov     eax, offset CheckForFEClockStateMsg
    cmp     eax, LastDebugMessage
    je      CheckForFEClockStateNomsg
    mov     eax, LastDebugMessage
    push    eax
    push    DPCScreen
    call    OutputToScreen
    lea     esp, [esp + (2 * 4)]
```

CheckForFEClockStateNomsg:

```
    mov     edx, [ebp].IOStatusAddr
    in      al, dx
    and     al, FEC_LOCK_MASK
    je      CheckFECNotLocked
```

```
    mov     [ebp].DemodStatus, LOCKED
    mov     [ebp].FecStatus, LOCKED
```

```
    mov     edx, [ebp].IOGateCounterHghAddr
    xor     eax, eax
    out     dx, al
```

```
    mov     edx, [ebp].IOCounterDeltaAddr
    mov     eax, [ebp].ReacqDeltaCounter
    out     dx, al
```

```
    mov     edx, [ebp].IOBtrControlAddr
    in      al, dx
    and     al, NOT (FREQ_PWR_MASK OR PHASE_PWR_MASK)
    out     dx, al
```

```
    in      al, dx
    or      al, 5
    out     dx, al
```

```
    mov     [ebp].NextStepCounter, 1
    mov     [ebp].T1Counter, 500
    mov     [ebp].T2Counter, 100
    mov     [ebp].CurrentState, TRACKING
    ret
```

CheckFECNotLocked:

```
    cmp     [ebp].T1Counter, 0
    jne     CheckFECHaveT1Counter
```

```
    mov     edx, [ebp].IOStatusAddr
    in      al, dx
    test    al, CRL_LOCK_MASK
    jne     CheckFECSetOtherMode
```

```
    mov     [ebp].CurrentState, INIT
    ret
```

CheckFECSetOtherMode:

```
    mov     [ebp].CurrentState, SET_OTHER_MODE
    CheckFECExit:
    ret
```

CheckFECHaveT1Counter:

```
    mov     edx, [ebp].IOStatusAddr
    in      al, dx
```

```
test    al, CRL_LOCK_MASK
jne     CheckFECExit
mov     eax, [ebp].MaxSqr
cmp     eax, [ebp].SqrAvg
jbe     CheckFECExit
sub     eax, 2
mov     [ebp].MaxSqr, eax
mov     edx, [ebp].IOctHaddr
out     dx, al
ret
```

```
CheckForFEClockState endp
subttl -- SetOtherModeState --
page
```

BEGIN_MANUAL_ENTRY(SetOtherModeState, DPC/API/SETOTHER)

Name: SetOtherModeState

Description: Acquisition State Routine.

```
On Entry:  EAX  N/A
           EBX  Frame Data Space
           ECX  N/A
           EDX  N/A
           EBP  Adapter Data Space
           ESI  N/A
           EDI  N/A
```

Note: Interrupts are in any state.

```
On Return: EAX  Destroyed
           EBX  Preserved
           ECX  Destroyed
           EDX  Destroyed
           EBP  Preserved
           ESI  Preserved
           EDI  Preserved
```

Flags:

Note: Interrupts preserved.

```
Remarks: This routine is called by DriverCallBack.
           It can be called at process or interrupt time.
```

See Also:

END_MANUAL_ENTRY

```
public SetOtherModeState
SetOtherModeState proc
```

```
    cmp     DebugMask, 0
    je      SetOtherModeStateNomsg
    mov     eax, offset SetOtherModeStateMsg
    cmp     eax, LastDebugMessage
    je      SetOtherModeStateNomsg
    mov     eax, LastDebugMessage
    push    eax
    push    DPCScreen
    call    OutputToScreen
```

```
lea esp, [esp + (2 * 4)]
SetOtherModeStateNomsg:
```

```
mov     edx, [ebp].IOSynthSerControlAddr
in      al, dx
cmp     [ebp].ViterbiMode, LOWRATE
jne     SetOtherModeToLow
```

```
or      al, MODE_MASK
out     dx, al
```

```
mov     [ebp].ViterbiMode, HIGHRATE
jmp     SetOtherModeIncrRate
```

```
SetOtherModeToLow:
and     al, NOT MODE_MASK
out     dx, al
```

```
mov     [ebp].ViterbiMode, LOWRATE
```

```
SetOtherModeIncrRate:
inc     [ebp].RateCount
```

```
cmp     [ebp].RateCount, 1
jg      SetOtherModeExit
```

```
mov     edx, [ebp].IOSynthSerControlAddr
in      al, dx
or      al, RESET_FEC_ACQ_MASK
out     dx, al
```

```
mov     [ebp].TlCount, 180
```

```
mov     edx, [ebp].IOSynthSerControlAddr
in      al, dx
and     al, NOT RESET_FEC_ACQ_MASK
out     dx, al
```

```
mov     [ebp].CurrentState, CHECK_FOR_FEC_LOCK
ret
```

```
SetOtherModeExit:
mov     [ebp].CurrentState, INIT
ret
```

```
SetOtherModeState     endp
subttl  -- ReadWord --
page
```

```
.....\
; BEGIN_MANUAL_ENTRY( ReadWord, DPC/API/READWORD )
```

```
; Name:      ReadWord
```

```
; Description: Acquisition State Routine.
```

```
; On Entry:  EAX      N/A
```

```
           EBX      Frame Data Space
```

```
           ECX      N/A
```

```
           EDX      N/A
```

```
           EBP      Adapter Data Space
```

```
           ESI      N/A
```

```
           EDI      N/A
```

```
; Note:      Interrupts are in any state.
```

```
; On Return:
```

```
      EAX      Destroyed
      EBX      Preserved
      ECX      Destroyed
      EDX      Destroyed
      EBP      Preserved
      ESI      Preserved
      EDI      Preserved
```

```
Flags:
```

```
Note:      Interrupts preserved.
```

```
Remarks:
```

```
      This routine is called by TrackingState,
      PointingAcquisitionState and PointingTrackingState
      It can be called at process or interrupt time.
```

```
; See Also:
```

```
; END_MANUAL_ENTRY
```

```
...../
; public ReadWord
; proc
```

```
ReadWord     push     ebx
```

```
             xor     ebx, ebx
```

```
             mov     edx, ebx
```

```
             in      al, dx
```

```
             mov     bh, al
```

```
ReadWordLoop: mov     ecx, 4
```

```
             mov     edx, esi
```

```
             in      al, dx
```

```
             mov     bl, al
```

```
             mov     edx, edi
```

```
             in      al, dx
```

```
             cmp     al, bh
```

```
             je      ReadWordExit
```

```
             mov     bh, al
```

```
             dec     ecx
```

```
             jne     ReadWordLoop
```

```
ReadWordExit: mov     eax, ebx
```

```
             pop     ebx
```

```
             ret
```

```
ReadWord     subttl  -- TrackingState --
```

```
             page
```

```
.....\
; BEGIN_MANUAL_ENTRY( TrackingState, DPC/API/TRCKST )
```

```
; Name:      TrackingState
```

```
; Description: Acquisition State Routine.
```

```
; On Entry:    EAX    N/A
               EBX    Frame Data Space
               ECX    N/A
               EDX    N/A
               EBP    Adapter Data Space
               ESI    N/A
               EDI    N/A

; Note:    Interrupts are in any state.

; On Return:  EAX    Destroyed
               EBX    Preserved
               ECX    Destroyed
               EDX    Destroyed
               EBP    Preserved
               ESI    Preserved
               EDI    Preserved

; Flags:

; Note:    Interrupts preserved.

; Remarks:    This routine is called by DriverCallback
               It can be called at process or interrupt time.

; See Also:

; END_MANUAL_ENTRY
;...../

public TrackingState
TrackingState proc

    cmp     DebugMask, 0
    je      TrackingStateNomsg
    mov     eax, offset TrackingStateMsg
    cmp     eax, LastDebugMessage
    je      TrackingStateNomsg
    mov     eax, LastDebugMessage
    push    eax
    push    DPCScreen
    push    OutputToScreen
    call    esp, (esp + (2 * 4))
    TrackingStateNomsg:

    mov     edx, [ebp].IOStatusAddr
    in      al, dx
    test    al, FEC_LOCK_MASK
    jne     TrackingStateReadQuality

    in      al, dx
    test    al, CRL_LOCK_MASK
    je      TrackingStateZero
    cmp     [ebp].T2Count, 0
    jne     TrackingStateT1

TrackingStateZero:
    mov     [ebp].TrackingMode, 0
    mov     [ebp].CurrentState, INIT
    ret

TrackingStateT1:
    mov     [ebp].T1Count, 1000
    ret
```

```
TrackingStateReadQuality:
    xor     eax, eax
    mov     edx, [ebp].IORefSeqAddr
    in      al, dx
    mov     [ebp].SignalQuality, eax

    cmp     DebugMask, 0
    je      SignalStrengthNomsg
    cmp     LastSignalStrength, 0
    jne     SignalStrengthNomsg

    mov     LastSignalStrength, 1
    cmp     eax, 200
    jb      SignalStrengthNone
    sub     eax, 200
    shl     eax, 1
    add     eax, 60
    jmp     SignalStrengthPrint

SignalStrengthNone:
    xor     eax, eax
SignalStrengthPrint:
    push    eax
    push    offset SignalStrengthMsg
    push    DPCScreen
    call    OutputToScreen
    lea     esp, [esp + (3 * 4)]

SignalStrengthNomsg:
    cmp     [ebp].T1Count, 0
    jne     TrackingStateExit

    mov     [ebp].T1Count, 1000
    mov     [ebp].TrackingMode, 1

    mov     edi, [ebp].IOtuningHighAddr
    mov     esi, [ebp].IOtuningLowAddr
    call    ReadWord
    mov     [ebp].Drift, eax

    mov     ecx, 2
    cmp     eax, NOW_COUNT_TRACK + OFFSET_THRESHOLD
    ja      TrackingStateLocFound
    mov     ecx, 0
    cmp     eax, NOW_COUNT_TRACK - OFFSET_THRESHOLD
    jnb     TrackingStateLocFound
    mov     ecx, 1
    TrackingStateLocFound:
    mov     [ebp].SearchLoc, ecx
    mov     [ebp].SearchLocFound, TRUE
TrackingStateExit:
    ret

TrackingState    endp
subctl -- PointingAcquisitionState --
page
```

```
.....\
; BEGIN_MANUAL_ENTRY ( PointingAcquisitionState, DPC/API/PTACOST )
; Name:    PointingAcquisitionState
; Description: Acquisition State Routine.
; On Entry:    EAX    N/A
```

Frame Data Space

N/A

N/A

Adapter Data Space

N/A

N/A

Note: Interrupts are in any state.

On Return:

EAX Destroyed

EBX Preserved

ECX Destroyed

EDX Destroyed

EBP Preserved

ESI Preserved

EDI Preserved

Flags: Interrupts preserved.

Remarks: This routine is called by DriverCallBack.

It can be called at process or interrupt time.

See Also:

END_MANUAL_ENTRY

public PointingAcquisitionState
PointingAcquisitionState proc

cmp DebugMask, 0

je PointingAcquisitionStateNomsg

mov eax, offset PointingAcqStateMsg

cmp eax, LastDebugMessage

je PointingAcquisitionStateNomsg

mov LastDebugMessage, eax

push DPCScreen

push OutputToScreen

lea esp, [esp + (2 * 4)]

PointingAcquisitionStateNomsg:

mov edx, [ebp].IOStatusAddr

in al, dx

test al, SWEEPING_MASK

je PointingNotSweeping

mov esi, [ebp].IOTuningLowAddr

mov edi, [ebp].IOTuningHighAddr

call ReadWord

shl eax, 4

mov [ebp].Drift, eax

mov [ebp].DemodStatus, LOCKED

xor eax, eax

mov edx, [ebp].IOGateCounthighAddr

out dx, al

mov edx, [ebp].IOBtrControlAddr

in al, dx

and al, NOT (FREQ_PWR_MASK OR PHASE_PWR_MASK)

out dx, al

in al, dx

or al, 5

out dx, al

mov [ebp].NextStepCount, 1

mov [ebp].r1Count, 1000

mov [ebp].SearchLocFound, FALSE

mov [ebp].CurrentState, POINTING_TRACKING

ret

PointingNotSweeping:

mov eax, [ebp].MaxSqt

sub eax, 2

mov [ebp].MaxSqt, eax

mov edx, [ebp].IOCchAddr

out dx, al

cmp [ebp].r1Count, 0

jne PointingAcqExit

mov [ebp].CurrentState, INIT

PointingAcqExit:

ret

PointingAcquisitionState endp

subttl -- PointingTrackingState --

page

Name: PointingTrackingState

Description: Acquisition State Routine.

On Entry:

EAX N/A

EBX Frame Data Space

ECX N/A

EDX N/A

EBP Adapter Data Space

ESI N/A

EDI N/A

Note: Interrupts are in any state.

On Return:

EAX Destroyed

EBX Preserved

ECX Destroyed

EDX Destroyed

EBP Preserved

ESI Preserved

EDI Preserved

Flags:

Note: Interrupts preserved.

Remarks: This routine is called by DriverCallBack.

It can be called at process or interrupt time.

See Also:

END_MANUAL_ENTRY

```

; .....
; public PointingTrackingState
; PointingTrackingState proc

```

```

    cmp     DebugMask, 0
    je      PointingTrackingStateNomsg
    mov     eax, offset PointingTrackingStateMsg
    cmp     eax, LastDebugMessage
    je      PointingTrackingStateNomsg
    mov     eax, LastDebugMessage
    push    eax
    push    DPCScreen
    call    OutputToScreen
    lea     esp, [esp + (2 * 4)]
    PointingTrackingStateNomsg:
    xor     eax, eax
    mov     edx, [ebp].IOrelsqfAddr
    in      al, dx
    mov     [ebp].SignalQuality, eax

```

```

    cmp     [ebp].TICount, 0
    je      PointingTrackingExit

```

```

    mov     [ebp].TICount, 1000

```

```

    mov     esi, [ebp].IOTuningLowAddr
    mov     edi, [ebp].IOTuningHighAddr
    call    ReadWord

```

```

    mov     ecx, [ebp].Drift
    add     ecx, OFFSET_THRESHOLD
    cmp     eax, ecx
    je      PointingTrackingInit

```

```

    mov     ecx, [ebp].Drift
    sub     ecx, OFFSET_THRESHOLD
    cmp     eax, ecx
    jae     PointingTrackingExit

```

```

PointingTrackingInit:
    mov     [ebp].CurrentState, INIT

```

```

PointingTrackingExit:
    ret

```

```

PointingTrackingState endp
subttl -- HaltState --
page

```

```

; .....
; BEGIN_MANUAL_ENTRY( HaltState, DPC/API/HALTST )

```

```

; Name:      HaltState

```

```

; Description: Acquisition State Routine.

```

```

; On Entry:

```

```

    EAX    N/A
    EBX    Frame Data Space
    ECX    N/A
    EDX    N/A
    EBP    Adapter Data Space
    ESI    N/A
    EDI    N/A

```

```

; Note:      Interrupts are in any state.

```

```

; On Return:
; EAX      Destroyed
; EBX      Preserved
; ECX      Destroyed
; EDX      Destroyed
; EBP      Preserved
; ESI      Preserved
; EDI      Preserved

```

```

; Flags:

```

```

; Note:      Interrupts preserved.

```

```

; Remarks:   This routine is called by DriverCallback.
;            It can be called at process or interrupt time.

```

```

; See Also:

```

```

; END_MANUAL_ENTRY
; .....

```

```

; public HaltState
; HaltState proc

```

```

    cmp     DebugMask, 0
    je      HaltStateNomsg
    mov     eax, offset HaltStateMsg
    cmp     eax, LastDebugMessage
    je      HaltStateNomsg
    mov     eax, LastDebugMessage
    push    eax
    push    DPCScreen
    call    OutputToScreen
    lea     esp, [esp + (2 * 4)]
    HaltStateNomsg:
    ret

```

```

HaltState endp
subttl -- InitDemod --
page

```

```

; .....
; BEGIN_MANUAL_ENTRY( InitDemod, DPC/API/INITDMOD )

```

```

; Name:      InitDemod

```

```

; Description: Acquisition State Routine.

```

```

; On Entry:

```

```

    EAX    N/A
    EBX    Frame Data Space
    ECX    N/A
    EDX    N/A
    EBP    Adapter Data Space
    ESI    N/A
    EDI    N/A

```

```

; Note:      Interrupts are in any state.

```

```

; On Return:

```

```

    EAX    Destroyed
    EBX    Preserved
    ECX    Destroyed
    EDX    Destroyed
    EBP    Preserved

```

ESI Preserved
EDI Preserved

Flags:

Note: Interrupts preserved.

Remarks: This routine is called by DriverCallback.
It can be called at process or interrupt time.

See Also:

END_MANUAL_ENTRY

public InitDemod
proc

```
mov [ebp].CurrentState, HALT
mov [ebp].RxFreq, 0
mov [ebp].ViterbiMode, 0
mov [ebp].DemodCommand, HALT_MODE
mov [ebp].SearchLoc, 1
mov [ebp].Drift, 0
mov [ebp].GLOffset, 0
mov [ebp].TrackingMode, FALSE
```

```
; value = read_bits (STATUS_ADDR, TUNER_TYPE_MASK);
xor eax, eax
mov edx, [ebp].IOStatusAddr
in al, dx
and al, TUNER_TYPE_MASK
mov cl, al
```

```
; value |= read_bits (UNIT_ID_ADDR, TUNER_TYPE_2_MASK);
mov edx, [ebp].IOUnitIDAddr
in al, dx
and al, TUNER_TYPE_2_MASK
or al, cl
```

cmp al, SHARP
jne InitDemodPanasonic

```
cmp DebugMask, 0
je FillIntTunerVars
push eax
push offset SharpTunerMsg
push DPCScreen
call OutputToScreen
lea esp, [esp + (2 * 4)]
pop eax
jmp FillIntTunerVars
```

InitDemodPanasonic:
cmp al, PANASONIC
jne InitDemodSharpCustom

```
cmp DebugMask, 0
je FillIntTunerVars
push eax
push offset PanasonicTunerMsg
push DPCScreen
call OutputToScreen
lea esp, [esp + (2 * 4)]
```

pop eax
jmp FillIntTunerVars

InitDemodSharpCustom:

```
cmp al, SHARP_CUSTOM
jne InitDemodExit
```

```
cmp DebugMask, 0
je FillIntTunerVars
push eax
push offset SharpCustomTunerMsg
push DPCScreen
call OutputToScreen
lea esp, [esp + (2 * 4)]
pop eax
```

FillIntTunerVars:

```
mov [ebp].TunerTypeFound, eax
mov [ebp].ReacqGateCount, 0f0h
mov [ebp].ReacqDeltaCount, 2
mov [ebp].NomCountSearch, NOM_COUNT_REACQ - 75
mov [ebp].SqfCheckPoints, 11
mov [ebp].SqfCheckStepsSize, 15
mov [ebp].SqfDeltaCount, 8
```

InitDemodExit:
ret

InitDemod
subttl -- ApplyDelay --
page

BEGIN_MANUAL_ENTRY (ApplyDelay, DPC/API/APPLYDEL)

Name: ApplyDelay

Description: Acquisition State Routine.

On Entry:

EAX	N/A
EBX	Frame Data Space
ECX	N/A
EDX	N/A
EBP	Adapter Data Space
ESI	N/A
EDI	N/A

Note: Interrupts are in any state.

On Return:

EAX	Destroyed
EBX	Preserved
ECX	Destroyed
EDX	Destroyed
EBP	Preserved
ESI	Preserved
EDI	Preserved

Flags:

Note: Interrupts preserved.

Remarks:

This routine is called by DriverCallback.
It can be called at process or interrupt time.

See Also:

; END_MANUAL_ENTRY

;
; public ApplyDelay
; proc

; Apply delay to T1 Counter

```
cmp [ebp].T1Count, 0
je ApplyDelayT2
cmp [ebp].T1Count, eax
jb ApplyDelayClearT1
sub [ebp].T1Count, eax
jmp ApplyDelayT2
ApplyDelayClearT1:
mov [ebp].T1Count, 0
```

; Apply delay to T2 Counter

```
ApplyDelayT2:
cmp [ebp].T2Count, 0
je ApplyDelayExit
cmp [ebp].T2Count, eax
jb ApplyDelayClearT2
sub [ebp].T2Count, eax
jmp ApplyDelayExit
ApplyDelayClearT2:
mov [ebp].T2Count, 0
```

ApplyDelayExit:

ret

ApplyDelay endp

subttl -- CalculateRxFreq --

page

; BEGIN_MANUAL_ENTRY(CalculateRxFreq, DPC/API/CALCRXFQ)

; Name: CalculateRxFreq

; Description: Acquisition State Routine.

```
; On Entry: EAX N/A
; EBX Frame Data Space
; ECX N/A
; EDX N/A
; EBP Adapter Data Space
; ESI N/A
; EDI N/A
```

; Note: Interrupts are in any state.

; On Return:

```
EAX Destroyed
EBX Preserved
ECX Destroyed
EDX Destroyed
EBP Preserved
ESI Preserved
EDI Preserved
```

; Flags:

; Note: Interrupts preserved.

; Remarks: This routine is called by DriverCallback.
; It can be called at process or interrupt time.

; See Also:

; END_MANUAL_ENTRY

;
; public CalculateRxFreq
; CalculateRxFreq proc

```
; USHORT_T freq, total, new_total, results:
sub esp, 2 * 4
```

```
; total = [esp + 0]
; results = [esp + 4]
; freq = esi
; new_total = edi
```

; freq = S.Rx.Freq - FREQ_BASE;

```
; freq += S.GL_offset;
mov esi, [ebp].RxFreq
sub esi, FREQ_BASE
add esi, [ebp].GLOffset ; ESI = freq
```

; total = (freq * 2) / 729;

```
mov eax, esi ; EAX = freq
shl eax, 1 ; EAX = freq * 2
xor edx, edx
mov ecx, 729
div ecx ; EAX = EAX / 729
mov [esp + 0], eax ; total = EAX
```

; results = (total * 729) / 2;

```
mov ecx, 729 ; EAX = total * 729
mul ecx ; EAX = EAX * 729
shr eax, 1 ; results = eax
mov [esp + 4], eax ; results = eax
```

; freq = freq - results;

sub esi, eax

; new_total = total * 10;

```
mov eax, [esp + 0]
mov ecx, 10 ; EAX = total * 10
mul ecx ; new_total = EAX
mov edi, eax
```

; total = (freq * 20) / 729;

```
mov eax, esi ; EAX = freq
mov ecx, 20 ; EAX = freq * 20
mul ecx
xor edx, edx
mov ecx, 729 ; EAX = EAX / 729
div ecx ; total = EAX
mov [esp + 0], eax
```

; results = (total * 729) / 20;

```
mov ecx, 729 ; EAX = total * 729
mul ecx
xor edx, edx
mov ecx, 20 ; EAX = EAX / 20
div ecx
```



```
cmp [ebp].TrackingMode, FALSE
je CheckRxFreqSetMode
call CalculateRxFreq
CheckRxFreqSetMode:
mov [ebp].DemodCommand, ACQUIRE_MODE
; Possibly set the CurrentState depending on DemodCommand
;
; CallBackCheckState:
cmp [ebp].DemodCommand, ACQUIRE_MODE
je CallBackSetInit
cmp [ebp].DemodCommand, POINTING_MODE
je CallBackCheckHalt
; CallBackSetInit:
mov [ebp].CurrentState, INIT
call CallBackApplyDelay
jmp CallBackWatchDog
; CallBackCheckHalt:
cmp [ebp].DemodCommand, HALT_MODE
je CallBackApplyDelay
mov [ebp].CurrentState, HALT
; Apply delay
;
; CallBackApplyDelay:
mov eax, 15
call ApplyDelay
mov [ebp].CurrentState
mov esi, StateTbl[eax * 4]
call esi
; CallBackWatchDog:
ret
```

```
DriverCallBack endp
public DriverSend
subttl -- DriverSend --
page
*****
; BEGIN_MANUAL_ENTRY( DriverSend, DPC/API/SEND )
;
; Name: DriverSend
;
; Description: This routine will transfer the packet described in the
; TCB to the NIC and initiate the send. TxStartTime and
; RetryCounter must be set to enable the deadman timer.
;
; On Entry:
; EAX N/A
; EBX @ Frame Data Space
; ECX Padded Packet Length
; EDX N/A
; EBP @ Adapter Data Space
; ESI @ TCB
; EDI N/A
;
; Note: Interrupts are disabled.
;
; On Return:
; EAX Destroyed
; EBX Preserved
; ECX Destroyed
; EDX Destroyed
; EBP Preserved
; ESI Destroyed
; EDI Destroyed
;
; Flags:
; Note: Interrupts disabled.
;
; Remarks: This routine is called by the MSM media module.
; It is called at process or interrupt time.
;
; See Also:
; ETHERTSM\EtherTSM\DriverSend
; ETHERTSM\MediaSendRaw8023
; ETHERTSM\MediaSendEthernetII
; ETHERTSM\MediaSend8022Over8023
; ETHERTSM\MediaSend8022Snap
;
; END_MANUAL_ENTRY
;
; *****
;
; align 16
; DriverSend proc
;
; lea edi, [esi].TCBMediaHeader ; ARP(0x08 0x06)?
; cmp word ptr [edi+12], 0608h ; Jump if it is
; je DriverSendArp
;
; cmp [ebp].AgentSendRoutine, 0 ; Can we send it yet?
; je DriverSendExit
;
; t back if we can't
;
; push ecx
; push esi
;
; Padded size
; push esi
;
; Address of TCB
; call [ebp].AgentSendRoutine ; Give it to Slip Handlerr
```

```

pop     esi
pop     ecx
ret

```

```

; DriversSendExit:
; inc [ebp].MSMTxFreeCount ; Add a send resource
; jmp EthernetSMFastSendComplete ; Otherwise service events.

```

```

; DriversSendARP:

```

```

; We're going to assume that the entire request is in the first
; fragment. Verify it first.

```

```

mov     edi, [esi].TCBFragStructPtr ; One fragment?
cmp     dword ptr [edi+0], 1
jne     DriversSendExit ; Jump if not
cmp     dword ptr [edi+8], 28 ; Entire ARP request (frag length)?
jnb     DriversSendExit ; Jump if not

```

```

; Make sure sender and target ip addr are different

```

```

mov     edi, [edi+4] ; EDI -> ARP request
mov     eax, [edi+28-14] ; EAX = senders IP
cmp     eax, [edi+38-14] ; Same as target IP?
je      DriversSendExit ; Jump out if it is

```

```

push    esi ; Save send ECB
push    edi ; Save ARP offset

```

```

mov     esi, 1514 ; Max ECB size.
call    MSMAAllocateRCB ; Get an ECB
pop     edi ; EDI -> ARP request again
or      eax, eax ;
jne     DriversSendReturnARP ; Jump if no ECB.

```

```

; ESI -> reply ECB
; EDI -> request data

```

```

lea     eax, [esi+RPacketEnvelope] ; EAX -> beginning
mov     [esi].RPacketOffset, eax ; Store into ECB
mov     [esi].RPacketSize, 60 ; ARP reply size
mov     [esi].RPacketLength, 60 ; (ethernet min size)

```

```

push    esi ; Save reply ECB
mov     esi, eax ; ESI -> reply data

```

```

; ESI -> reply data
; EDI -> request data

```

```

; Set reply->dest_addr to our node address

```

```

mov     eax, dword ptr [ebx].MLIDNodeAddress+0
mov     dword ptr [esi+0], eax
mov     ax, word ptr [ebx].MLIDNodeAddress+4
mov     word ptr [esi+4], ax

```

```

; Set reply->source_addr to (0x06 0x06 0x06 0x06 0x06 0x06)

```

```

mov     word ptr [esi+6], 0606h
mov     dword ptr [esi+8], 06060606h

```

```

; Set reply->ctype to (0x08 0x06)

```

```

mov     word ptr [esi+12], 0606h

```

```

; Set reply hardware type(0x00 0x01), protocol type(0x08 0x00)

```

```

mov     dword ptr [esi+14], 00080100h

```

```

; Set reply hardware size(0x06), protocol size(0x04)
; and operation(0x00 0x02 for ARP reply)

```

```

mov     dword ptr [esi+18], 02000406h

```

```

; Set reply senders ethernet addr to (0x06 0x06 0x06 0x06 0x06 0x06).

```

```

mov     dword ptr [esi+22], 06060606h
mov     word ptr [esi+26], 0606h

```

```

; Set reply senders ip addr to the request target ip addr

```

```

mov     eax, [edi+38-14] ; request->target_ip
mov     [esi+28], eax

```

```

; Set reply target ethernet addr to our node addr

```

```

mov     eax, dword ptr [ebx].MLIDNodeAddress+0
mov     dword ptr [esi+32], eax
mov     ax, word ptr [ebx].MLIDNodeAddress+4
mov     word ptr [esi+36], ax

```

```

; Set reply target ip addr to request senders ip addr

```

```

mov     eax, dword ptr [edi+28-14] ; request->senders_ip
mov     dword ptr [esi+38], eax

```

```

pop     esi ; ESI -> reply ECB
mov     edi, 1514 ; Max Packet size
xor     eax, eax ; Good packet
mov     ecx, [esi].RPacketSize
push    ebp
call    EthernetSMFastProcessGetRCB

```

```

pop     ebp ; Jump if no new ECB
jne     DriversSendReturnARP ; Return newly allocated ECB
MSMReturnRCB

```

```

; DriversSendReturnARP:

```

```

pop     esi ; Restore send ECB
inc     [ebp].MSMTxFreeCount ; Add a send resource
jmp     EthernetSMFastSendComplete ; Otherwise service events.

```

```

; DriversSend

```

```

extrn   DoEndOfInterrupt: near
extrn   SetHardwareInterrupt: near
extrn   ClearHardwareInterrupt: near
proc    TestDriverISR

```

```

mov     ebp, OurAdapterDataSpace
mov     ebx, [ebp].MSMDefaultVirtuaBoard
movzx   ecx, [ebx].MLIDInterrupt
call    DoEndOfInterrupt

```

```

inc     [ebp].GotInterrupt

```

```

xor     eax, eax
ret

```

```

TestDriverISR endp

```

subject -- DriverISR --
page

BEGIN_MANUAL_ENTRY (DriverISR, DPC/API/ISR)

Name: DriverISR

Description: This routine handles packet reception.

On Entry:

EAX N/A
EBX N/A
ECX N/A
EDX N/A
EBP @ Adapter Data Space
ESI N/A
EDI N/A

Note: Interrupts are disabled.

On Return:

EAX Destroyed
EBX Destroyed
ECX Destroyed
EDX Destroyed
EBP Destroyed
ESI Destroyed
EDI Destroyed

Flags:

Note: Interrupts disabled.

Remarks:

This routine is called by the MSM.
It is called at interrupt time.

See Also: MSM\MSMInterruptProcedure

END_MANUAL_ENTRY

.....

align 16

public DriverISR

DriverISR

/* Set the adapters ram ptr to the next rbd to receive from */

output(bicd_base_addr + MSG_RAM_PTR, rbd_base_addr + 2*curr_adap_rbd);

DebugMessage DEBUG_ISR_ALL, ISREnterMsg

mov edx, [ebp].IOMsgRamPtr ; MsgRamPtr I/O port

mov eax, [ebp].CurrentAdapterRBD ; Next Adapter RBD

shl eax, 1 ; * 2

add eax, RBD_BASE_ADDR ; add Base(0a000h)

out dx, ax ; Set Adapter Ram Ptr

/* Keep processing packets until no more are left */

/*

NOTE: We are assuming that anyone looping back to DriverISRLoop

has set the MSR_RAM_PTR to the next RBD to examine.

*/

while ((status = import (bicd_base_addr + MSG_RAM) & EMPTY)

DriverISRLoop):

xor eax, eax

; Clear upper status

mov edx, [ebp].IOMsgRam
in ax, dx
mov [ebp].IntStatus, eax

test eax, EMPTY

je DriverISRExit

inc [ebp].BufferCount

DebugMessage1 DEBUG_ISR, DebugRBDReceived, [ebp].CurrentAdapterRBD

/* Jump if this is an error packet */

if (status & 0x8f)

test [ebp].IntStatus, STATUS_ERROR

jne DriverISRBadPacket

/* Here's a good packet. See if we have a buffer for it. */

if (!global_pool[curr_rbd].buf_ptr)

mov esi, [ebp].CurrentECB

or esi, esi

jne DriverISRAddToECB

if TIMESTAMP

mov al, 'r'

push eax

call DPCrTimestamp

lea esp, [esp + 4]

endif

mov esi, 1514

call MSMAllocaterCB

or eax, eax

jne DriverISRNoECB

!!! Satellite header is 12 bytes, EIP is 14 bytes.

!!! Add 2 to offset to prevent double copy of turbo internet packets.

lea edi, [esi+RPacketEnvelope*2] ; EDI -> beginning

mov [esi].RPacketOffset, edi ; Store into ECB

mov [esi].RPacketSize, 0 ; Clear size

mov [ebp].CurrentECB, esi ; Store if split packet

jmp short DriverISRReadSize

DriverISRAddToECB:

mov edi, [esi].RPacketOffset

add edi, [esi].RPacketSize

DriverISRReadSize:

/* ESI(curr_rbd) will be used a lot. Let's try to keep it intact. */

/* Retrieve the length of the packet */

length = import(bicd_base_addr + MSG_RAM);

xor eax, eax

mov edx, [ebp].IOMsgRam

in ax, dx

add [esi].RPacketSize, eax

DebugMessage1 DEBUG_ISR, DebugRBDSize, eax

word_length = (length & 3) ? (length / 4) + 1 : (length/4);

;

;

;

;

;

;

;

;

;

;

;

;

;

;

;

;

;

;

;

;

;

;

;

```

mov     ecx, eax
shr     ecx, 2
test    eax, 3
jz      DriverISRCopyPacket
inc     ecx

;; shr     ecx, 1
;; test   eax, 1
;; jz     DriverISRCopyPacket
inc     ecx
DriverISRCopyPacket:
    outport(bicd_base_addr + AUTO_INC, curr_adap_rbd * stats.config.buf_size /
2);
    push  ecx
    mov   eax, [ebp].CurrentAdapterRBD
    mov   ecx, RBD_BUFFER_SIZE
    mov   ecx, RBD_BUFFER_SIZE / 2
    mul   ecx
    shr   eax, 1
    mov   edx, [ebp].IOAutoInc
    out   dx, ax
    pop   ecx

    Read from RxData port really fast(this was inline assembly)

    cld
    mov   edx, [ebp].IORxData
    rep   insw
    rep   insw

    /* Release an adapter buffer */
    outport(bicd_base_addr + MSG_RAM_PTR, rbd_base_addr + 2 * curr_adap_rbd);
    outport(bicd_base_addr + MSG_RAM, 0);
    outport(bicd_base_addr + MSG_RAM, stats.config.buf_size);

    mov   edx, [ebp].IOmsgRamPtr
    mov   eax, [ebp].CurrentAdapterRBD
    shl   eax, 1
    add   eax, RBD_BASE_ADDR
    out   dx, ax

    mov   edx, [ebp].IOmsgRam
    xor   eax, eax
    out   dx, ax

    mov   eax, RBD_BUFFER_SIZE
    out   dx, ax

    /* Bump Current Adapter RBD value and make sure it wraps */
    if (++curr_adap_rbd == stats.config.adap_rbd_num)
        curr_adap_rbd = 0;

    mov   eax, [ebp].CurrentAdapterRBD
    inc   eax
    cmp   eax, ADAP_RBD_NUM
    jbp   DriverISRBDWrap
    xor   eax, eax
    DriverISRBDWrap:
    mov   [ebp].CurrentAdapterRBD, eax
    DebugMessage1  DEBUG_ISR_ALL, AdapterRBDMsg, eax

; ECX = packet length
; ECX = ECX / 4
; Any left over?
; Jump if not
; read another dword

; Now set ram ptr to next rbd in case we loop up to DriverISRLoop */
outport(bicd_base_addr + MSG_RAM_PTR, rbd_base_addr + 2*curr_adap_rbd);

    mov   edx, [ebp].IOmsgRamPtr
    shl   eax, 1
    add   eax, RBD_BASE_ADDR
    out   dx, ax

    /* This was the first buffer. Save it off for unravel */
    first_buffer_rbd = curr_rbd;
    first_buffer = 0;
    frame_buffers = 0;

    if (!(status & EOF_BIT))
    {
        test [ebp].IntStatus, EOF_BIT
        je   DriverISRLoop

        /* We have the last frame. Pass buffers to application. */
        /* First lets see if we can match the packet address to a
        * filter table address.
        */
        for (i1 = 0; i1 < BICDD_MAX_ADDR; i1++)
            if (!memcmp( filter[i1].address, global_pool[first_buffer_rbd].buf_ptr, 6
&& (filter[i1].channel != BICDD_NOT_USED)

            mov [ebp].CurrentECB, 0
            lea edi, [ebp].Filter
            DriverISFilterLoop:
            mov     eax, dword ptr [edi].FilterAddress+0
            word
            mov     edx, [esi].RPacketOffset
            cmp     eax, [edx+0]
            jne     DriverISRFilterNext
            mov     ax, word ptr [edi].FilterAddress+4
            cmp     ax, [edx+4]
            jne     DriverISRFilterNext

            /* We have a winner! */
            /* EDI points to Filter entry */
            mov     eax, [edi].FilterChannel
            cmp     eax, RBD_NOT_USED
            je      DriverISRFilterNext

            DebugMessage6  DEBUG_ISR_ALL, FilterAddrMsg, [edx+0], [edx+1], [edx+2],
[edx+3], [edx+4], [edx+5]

            /* Lets set ESI to Rx Control entry */
            cc = filter[i1].channel;

            lea     ebx, [ebp].RxControl[ecx*8]

            /* Check whether length in the header matches the length rxed */
            for (i1 = 0, length = 0, mm = first_buffer_rbd; i1 < frame_buffers; i1++)

```



```
mov     (ebp).LargestRx, eax

DebuggerAve:
inc     (ebp).NumberLargerX
add     eax, (ebp).TotalLargerX
mov     edi, (ebp).NumberLargerX
mov     (ebp).TotalLargerRx, eax
xor     edx, edx
div     edi
mov     (ebp).AveLargerRx, eax

DebuggerExit:
; *** DEBUG
;
;
; if TIMESTAMP
;   push     ecx
;   mov     al, 'R'
;   push     eax
;   call     DPCTimeStamp
;   lea     esp, [esp + 4]
;   pop     ecx
; endif
xor     eax, eax
; Good packet

push     ebp
call     EthernetSMFastProcessGetRCB
pop      ebp
jne     DriverISRLoop
; no ecb returned
; Jump 1
jmp     DriverISRDidntWantECB
; Give it back. We don't store it
hese

DriverISRNotOurs:
push     esi
call     ESR
pop      esi
or       eax, eax
je       DriverISRLoop
; Param0 = ECB
; call ESR
; clean up stack
; did they keep it?
; jump if they did.

DriverISRDidntWantECB:
MSMReturnRCB
jmp     DriverISRLoop
; Return it
; Get next packet

DriverISRFilterNext:
add     edi, size FilterStruct
lea     eax, (ebp).FilterIMAX_ADDR * size FilterStruct
cmp     edi, eax
jbe     DriverISRFilterLoop
; Couldn't find filter address. Clean up.

MSMReturnRCB
DebugMessage6   DEBUG_ISR_ALL, FilterNone, [edx+0], [edx+1], [edx+2], (e
dx+3], [edx+4], [edx+5]
jmp     DriverISRLoop
; Give ECB back.

DriverISRFilterSeqNoMatch:
inc     eax
inc     [edi].FilterSeqCount
mov     [edi].FilterSeqNum, eax
jmp     DriverISRFilterCallISR

DriverISRFilterNoFilterSeq:
inc     eax
mov     [edi].FilterSeqNum, eax
jmp     DriverISRFilterCallISR

DriverISRFilterNoPacketSeq:
mov     [edi].FilterSeqNum, 1
jmp     DriverISRFilterCallISR

DriverISRFilterSkipRBDLen:
MSMReturnRCB
DebugMessage2   DEBUG_ISR_ALL, FilterRBDLen, ecx, edx
jmp     DriverISRLoop

DriverISRNoECB:
DebugMessage   DEBUG_ISR, NoECBMsg
jmp     DriverISRBadNextRBD

DriverISRBadPacket:
mov     esi, [ebp].InstStatus
test    esi, FRAMING_ERR
je       DriverISRCheckAbort
;
; DebugMessage   DEBUG_ISR, FramingErrMsg

DriverISRCheckAbort:
test    esi, ABORT
je       DriverISRCheckAlign
;
; DebugMessage   DEBUG_ISR, AbortMsg

DriverISRCheckAlign:
test    esi, ALIGN_ERR
je       DriverISRCheckOverrun
;
; DebugMessage   DEBUG_ISR, AlignErrMsg

DriverISRCheckOverrun:
test    esi, OVERRUN_ERR
je       DriverISRCheckDES
;
; DebugMessage   DEBUG_ISR, OverrunErrMsg

DriverISRCheckDES:
test    esi, DES_ERR
je       DriverISRCheckCRC
;
; DebugMessage   DEBUG_ISR, DESErrMsg

DriverISRCheckCRC:
test    esi, CRC_ERR
je       DriverISRFilterStats
;
; DebugMessage   DEBUG_ISR, CRCErrMsg

DriverISRFilterStats:
DebugMessage   DEBUG_ISR, ReturnMsg

DriverISRBadNextRBD:
mov     edx, [ebp].IOMsgRamPtr
mov     eax, [ebp].CurrentAdapterRBD
shl     eax, 1
add     ecx, RBD_BASE_ADDR
out     dx, ax
```

```
mov     cdx, [ebp].IOMsgRam
xor     eax, eax
out     dx, ax

mov     eax, RBD_BUFFER_SIZE
out     dx, ax

mov     eax, [ebp].CurrentAdapterRBD
inc     eax
cmp     eax, ADAP_RBD_NUM
jnb     DriverISRBadRBDWrap
xor     eax, eax
DriverISRBadRBDWrap:
mov     [ebp].CurrentAdapterRBD, eax

DebugMessage! DEBUG_ISR_ALL, AdapterRBDMsg, eax
mov     edx, [ebp].IOMsgRamPtr
shl     eax, 1
add     eax, RBD_BASE_ADDR
out     dx, ax
jmp     DriverISRLoop

DriverISRExit:
mov     edx, [ebp].IOMsgRamPtr
mov     eax, 0c3a0h
out     dx, ax

mov     eax, [ebp].CurrentAdapterRBD
mov     edx, [ebp].IOMsgRam
out     dx, ax

DebugMessage! DEBUG_ISR_ALL, ISRExitMsg, eax

mov     edx, [ebp].IOStatus
in      ax, dx
ret

DriverISR     endp
subttl  -- DriverDisableInterrupt --
page

;.....\

BEGIN_MANUAL_ENTRY( DriverDisableInterrupt, DPC/API/DISINT )
;
; Name:      DriverDisableInterrupt
;
; Description: This routine will disable the adapters ability to
;              interrupt the host.
;
; On Entry:
;
;   EAX  N/A
;   EBX  N/A
;   ECX  N/A
;   EDX  N/A
;   EBP  0 Adapter Data Space
;   ESI  N/A
;   EDI  N/A
;
; Flags:
;
; Note:      Interrupts are disabled.
;
; On Return:
;
;   EAX  Destroyed
;   EBX  Preserved
;   ECX  Preserved
;   EDX  Destroyed
;   ESI  Preserved
;   EDI  Destroyed
;
; Remarks:   This routine is called by the MSM.
;
; See Also:  DriverDisableInterrupt
;
; END_MANUAL_ENTRY
;...../
```

```
EBP  Preserved
ESI  Preserved
EDI  Preserved

Flags:

Note:      Interrupts disabled.

Remarks:   This routine is called by the MSM.

See Also:  DriverEnableInterrupt

END_MANUAL_ENTRY
;...../

align 16
DriverDisableInterrupt proc
xor     eax, eax
ret

DriverDisableInterrupt endp
subttl  -- DriverEnableInterrupt --
page

;.....\

BEGIN_MANUAL_ENTRY( DriverEnableInterrupt, DPC/API/ENINT )
;
; Name:      DriverEnableInterrupt
;
; Description: This routine will enable the adapters ability to
;              interrupt the host.
;
; On Entry:
;
;   EAX  N/A
;   EBX  N/A
;   ECX  N/A
;   EDX  N/A
;   EBP  0 Adapter Data Space
;   ESI  N/A
;   EDI  N/A
;
; Note:      Interrupts are disabled.
;
; On Return:
;
;   EAX  Destroyed
;   EBX  Preserved
;   ECX  Preserved
;   EDX  Destroyed
;   EBP  Preserved
;   ESI  Preserved
;   EDI  Preserved
;
; Flags:
;
; Note:      Interrupts disabled.
;
; Remarks:   This routine is called by the MSM.
;
; See Also:  DriverDisableInterrupt
;
; END_MANUAL_ENTRY
;...../

align 16
```



```
DriverEnableInterrupt proc
    ret
DriverEnableInterrupt endp
    public DriverReset
    subttl -- DriverReset --
    page
    .....
    BEGIN_MANUAL_ENTRY( DriverReset, DPC/API/RESET )
; Name:      DriverReset
; Description: This routine will reset and initialize the NIC.
;
; On Entry:  EAX  N/A
;            EBX  0 Frame Data Space
;            ECX  N/A
;            EDX  N/A
;            EBP  0 Adapter Data Space
;            ESI  N/A
;            EDI  N/A
;
; Note:      Interrupts are disabled.
;
; On Return: EAX  0 if successful (otherwise points to error message)
;            EBX  Preserved
;            ECX  Destroyed
;            EDX  Destroyed
;            EBP  Preserved
;            ESI  Destroyed
;            EDI  Destroyed
;
; Flags:
;
; Note:      Interrupts disabled.
;
; Remarks:   This routine is called by the MSM media module.
;            It is called at process time.
;
; See Also:  ETHERTSM\EtherTSMReset
;
; END_MANUAL_ENTRY
; .....
DriverReset proc near
    inc [ebp].AdapterResetCount ; Increment stat counter.
    xor eax, eax
    ret
DriverReset endp
DefaultRxFram proc
    ret
DefaultRxFram endp
extrn LSLGetStackIDFromName: near
ProtocolBindEvent proc
    lea edx, IPName
    call LSLGetStackIDFromName ; Return Stack ID in EBX
    or eax, eax
    jne short ProtocolBindExit
    mov esi, [esp + Parm0]
    cmp [esi+4], ebx
    jne short ProtocolBindExit ; IP Stack?
    mov edx, [esi]
    mov ebp, OurAdapterDataSpace ; EDX = Bound board number
    xor ecx, ecx
    ProtocolUnbindLoop:
        mov ebx, [ebp+MSMVirtualBoardLink][ecx*4]
        or ebx, ebx
        jz ProtocolBindNext
        cmp [ebx].MLIDBoardNumber, dx
        jne ProtocolBindNext
        mov eax, 1514
        mov [ebx].MLIDMaximumSize, eax
        sub eax, 14
        mov [ebx].MLIDMaxRecvSize, eax
        mov [ebx].MLIDRecvSize, eax
        ProtocolBindNext:
            inc ecx
            cmp ecx, 4
            jb ProtocolBindLoop
    ProtocolBindExit:
        CPop
        ret
ProtocolBindEvent endp
ProtocolUnbindEvent proc
    CPush
    lea edx, IPName
    call LSLGetStackIDFromName ; Return Stack ID in EBX
    or eax, eax
    jne short ProtocolUnbindExit
    mov esi, [esp + Parm0]
    cmp [esi+4], ebx
    jne short ProtocolUnbindExit ; IP Stack?
    mov edx, [esi]
    mov ebp, OurAdapterDataSpace ; EDX = Bound Board Number
    xor ecx, ecx
    ProtocolUnbindLoop:
        mov ebx, [ebp+MSMVirtualBoardLink][ecx*4]
        or ebx, ebx
        jz ProtocolUnbindNext
        cmp [ebx].MLIDBoardNumber, dx
        jne ProtocolUnbindNext
        mov eax, 1494
        mov [ebx].MLIDMaximumSize, eax
        sub eax, 14
        mov [ebx].MLIDMaxRecvSize, eax
```

```
mov     [ebx].MLIDRecvSize, eax
ProtocolUnbindNext:
    inc     ecx
    cmp     ecx, 4
    jnb     ProtocolUnbindLoop
ProtocolUnbindExit:
    CPOD
    ret
```

```
ProtocolUnbindEvent     endp
```

```
subttl  -- DriverInit --
page
```

```
.....\
BEGIN_MANUAL_ENTRY( DriverInit, DPC/API/INIT )
.....\
```

```
Name:      DriverInit
```

```
Description: This routine will call EthernetRegisterHSM,
              MSMParseDriverParameters, MSMRegisterHardwareOptions,
              MSMSetHardwareInterrupt, MSMRegisterMLID, initialize
              variables in the Adapter Data Space and reset/initialize
              the card.
```

```
On Entry:   EAX     N/A
            EBX     N/A
            ECX     N/A
            EDX     N/A
            EBP     N/A
            ESI     N/A
            EDI     N/A
```

```
Note:      Interrupts are enabled.
```

```
On Return:  EAX     0 if successful (otherwise it points to error message)
            EBX     Preserved
            ECX     Destroyed
            EDX     Destroyed
            EBP     Preserved
            ESI     Preserved
            EDI     Preserved
```

```
Flags:
```

```
Note:      Interrupts preserved.
```

```
Remarks:   This routine is called by the OS at load time.
            It is called at process time.
```

```
See Also:
```

```
MSM\MSMParseDriverParameters
MSM\MSMRegisterHardwareOptions
MSM\MSMSetHardwareInterrupts
MSM\MSMRegisterMLID
MSM\MSMScheduleInterruptCallback
MSM\MSMScheduleLESCallback
MSM\MSMEnablePolling
DriverReset
```

```
END_MANUAL_ENTRY
.....\
.....\
```

```
extrn     RegisterForEventNotification: near
extrn     UnRegisterEventNotification: near
DriverInit     proc
```

```
    CPOD
    if TIMESTAMP
        lea     eax, DPCTB
        mov     timestamp_begin, eax
        mov     timestamp_index, eax
        add     eax, TIMESTAMP_BUFFER_SIZE
        mov     timestamp_end, eax
    endif
    .....\
```

```
    Fill in Driver Parameter Block fields.
    .....\
```

```
    mov     DriverStackPointer, esp
    lea     esi, DriverParameterBlock
    call    EthernetRegisterHSM
    jnz     DriverInitError
    ; Fill in stack ->.
    ; ESI -> Parm block.
    ; Get EBX.
    ; Jump if error.
```

```
    ; Yuck! We'll have to adjust the receive size down, since
    ; Hughes can't handle full 1500 byte packets with tunneling.
```

```
    mov     [ebx].MLIDMaximumSize, 1494
    .....\
    ; EBX -> Frame Data Space(Config Table).
    ; Let MSM Parse the command line.
    ; .....\
```

```
    mov     GlobalRxFreq, DEFAULT_RX_FREQ
    mov     eax, NeedsIOPort0Bit OR NeedsInterrupt0Bit OR CAN_SET_NODE_ADDRESS
    lea     ecx, AdapterOptions
    call    MSMParseDriverParameters
    jnz     DriverInitError
    ; Jump if error.
    ; .....\
```

```
    Let MSM Register the hardware options.
    ; .....\
```

```
    call    MSMRegisterHardwareOptions
    cmp     eax, 1
    ja     DriverInitError
    je     DriverInitExit
    ; Error Registering?
    ; Jump if so.
    ; Skip if new frame.
```

```
    mov     OurAdapterDataSpace, ebp
    DPCRxFrame, offset DefaultRxFrame
    Get a timer resource tag so that we can delay ourselves.
    ; Save for later
```

```
    push    TimerSignature
    push    offset TimerDesc
    push    DriverModuleHandle
    call    AllocateResourceTag
```

```

lea     esp, [esp + (3 * 4)]
mov     [ebp].TimerTag, eax
or      eax, eax
lea     eax, ErrorAllocatingRtagMessage
je      DriverInitErrorReturn

; Get a resource tag for interrupts.
push    InterruptSignature
push    offset InterruptRtagMessage
push    DriverModuleHandle
call    AllocateResourceTag
add     esp, (3 * 4)
mov     [ebp].ISRtag, eax
or      eax, eax
lea     eax, ErrorAllocatingRtagMessage
je      DriverInitErrorReturn

; Get a resource tag for event notification.
push    EventSignature
push    offset EventRtagMessage
push    DriverModuleHandle
call    AllocateResourceTag
add     esp, (3 * 4)
mov     [ebp].EventTag, eax
or      eax, eax
lea     eax, ErrorAllocatingRtagMessage
je      DriverInitErrorReturn

; Register for protocol bind event.
mov     eax, [ebp].EventTag
push    offset ProtocolBindEvent
push    0
push    EVENT_PRIORITY_OS
push    EVENT_PROTOCOL_UNBIND
push    eax
call    RegisterForEventNotification
add     esp, (4 * 5)
mov     [ebp].ProtocolBindID, eax
or      eax, eax
je      DriverInitNoBindEvent

mov     eax, [ebp].EventTag
push    offset ProtocolUnbindEvent
push    0
push    EVENT_PRIORITY_OS
push    EVENT_PROTOCOL_UNBIND
push    eax
call    RegisterForEventNotification
add     esp, (4 * 5)
mov     [ebp].ProtocolUnbindID, eax
or      eax, eax
je      DriverInitSetPorts

DriverInitNoBindEvent:
mov     eax, 1494
mov     [ebx].MLIMaximumSize, eax
sub     eax, 14
mov     [ebx].MLIMaxRecvSize, eax
mov     [ebx].MLIDRecvSize, eax

DriverInitSetPorts:
; *****

; Set and check the adapters base I/O.
; *****
movzx   ecx, [ebx].MLIDIOPortsAndLengths
mov     [ebp].IORxData, ecx

0:
add     ecx, 2
mov     [ebp].IOAutoInc, ecx
2:
add     ecx, 2
mov     [ebp].IOStatus, ecx
add     ecx, 2
mov     [ebp].IOControl, ecx
6:
add     ecx, 2
mov     [ebp].IOMsgRamPtr, ecx
add     ecx, 2
mov     [ebp].IOMsgRam, ecx
add     ecx, 2
mov     [ebp].IORdbBufLen, ecx
2:
add     ecx, 2
[ebp].IORdbNum, ecx
add     ecx, 2
[ebp].IOBtrControlAddr, ecx
e + 10h + 0h
add     ecx, 2
[ebp].IOAfcControlAddr, ecx
e + 10h + 2h
add     ecx, 2
[ebp].IOBitDetControlAddr, ecx
base + 10h + 4h
add     ecx, 2
[ebp].IOAgcFirControlAddr, ecx
base + 10h + 6h
add     ecx, 2
[ebp].IOCr1KThrLowAddr, ecx
se + 10h + 8h
add     ecx, 2
[ebp].IOCr1Addr, ecx
+ 0ah
add     ecx, 2
mov     [ebp].IOGateCountHighAddr, ecx
base + 10h + 0ch
add     ecx, 2
[ebp].IOCountNomLowAddr, ecx
ase + 10h + 0eh
add     ecx, 2
mov     [ebp].IOCountNomHighAddr, ecx
base + 10h + 10h
add     ecx, 2
[ebp].IOCountDeltaAddr, ecx
e + 10h + 12h
add     ecx, 2
[ebp].IOSweepRateAddr, ecx
+ 10h + 14h
add     ecx, 2
[ebp].IOCr1kControlAddr, ecx
se + 10h + 16h
add     ecx, 2
[ebp].IOSynthesizerControlAddr, ecx
= base + 10h + 18h
add     ecx, 2
[ebp].IOSpareIOControlAddr, ecx
; Rx Data port = base +
; Auto Inc port = base +
; Status port = base + 4
; Control port = base +
; Msg Ram Ptr = base + 8
; Msg Ram = base +
; RDB buf Len = base + 1
; RDB Num = base + 14
; BTR Control Addr = base
; AFC Control Addr = base
; Bit Det Control Addr =
; Agc Fir Control Addr =
; Cr1k Thr Low Addr = base
; Cth Addr = base + 10h
; Gate Count High Addr =
; Count Nom Low Addr = base
; Count Nom High Addr =
; Count Delta Addr = base
; Sweep Rate Addr = base
; Cr1k Control Addr = base
; Synthesizer Control Addr =
; Spare IO Control Addr =

```

```

; base + 10h + 1ah
add     ecx, 2
mov     [ebp].IODaOffsetSecControlAddr, ecx
ddr = base + 10h + 1ch
add     ecx, 2
mov     [ebp].IOUnitIDAddr, ecx
10h + 1eh
add     ecx, 2

movzx   ecx, [ebp].MLIDIOPortAndLengths
mov     al, 0bh
cmp     ecx, 100h
je      SetPort
dec     al
cmp     ecx, 140h
je      SetPort
dec     al
cmp     ecx, 180h
je      SetPort
dec     al
cmp     ecx, 1c0h
je      SetPort
dec     al
cmp     ecx, 200h
je      SetPort
dec     al
cmp     ecx, 240h
je      SetPort
dec     al
cmp     ecx, 280h
je      SetPort
dec     al
cmp     ecx, 2c0h
je      SetPort
dec     al
cmp     ecx, 300h
je      SetPort
dec     al
cmp     ecx, 340h
je      SetPort
dec     al
cmp     ecx, 380h
je      SetPort
dec     al
SetPort:
mov     dx, 279h
out     dx, al

; Lets Reset the adapter.
push    eax
mov     edx, [ebp].IOControl
mov     eax, CNTL_MRESET
out     dx, ax
mov     eax, [ebp].TimerTag
push    eax
push    2
call    DelayMyself
add     esp, (2 * 4)
mov     edx, [ebp].IOControl
mov     eax, 0
out     dx, ax

; IO DA Offset Control A
; Unit ID Addr = base +
; AL = 0bh
; I/O port 100h?
; yes
; AL = 0ah
; I/O port 140h?
; yes
; AL = 9
; I/O port 180h?
; yes
; AL = 8
; I/O port 1c0h?
; yes
; AL = 7
; I/O port 200h?
; yes
; AL = 6
; I/O port 240h?
; yes
; AL = 5
; I/O port 280h?
; yes
; AL = 4
; I/O port 2c0h?
; yes
; AL = 3
; I/O port 300h?
; yes
; AL = 2
; I/O port 340h?
; yes
; AL = 1
; I/O port 380h?
; yes
; AL = 0
; Set Base I/O port

; Make sure that we can set spare output
mov     edx, [ebp].IOControl
mov     eax, CNTL_SOUTPURT
out     dx, ax
; set Spare Output

; Make sure that we can clear spare output
mov     edx, [ebp].IOControl
mov     eax, 0
out     dx, ax
; Clear Spare Output

mov     edx, [ebp].IOStatus
in       ax, dx
test    eax, SPAT_SOUTPURT
mov     eax, offset MsGIOSetFailed
je      DriverInitErrorReturn

; Make sure that we can clear spare output
mov     edx, [ebp].IOControl
mov     eax, 0
out     dx, ax

mov     edx, [ebp].IOStatus
in       ax, dx
test    eax, SPAT_SOUTPURT
mov     eax, offset MsGIOClearFailed
jne      DriverInitErrorReturn

; If no node override, default it.
cmp     dword ptr [ebp].MLIDNodeAddress, -1
jnz     short NodeIsSet
mov     [ebp].MLIDNodeAddress+0, 00h
mov     [ebp].MLIDNodeAddress+1, 80h
mov     [ebp].MLIDNodeAddress+2, 0aeh
mov     [ebp].MLIDNodeAddress+3, 00h
mov     [ebp].MLIDNodeAddress+4, 00h
mov     [ebp].MLIDNodeAddress+5, 01h

NodeIsSet:
; Download the MIPS code to the adapter.
; *****
;
;
;
; Set to auto-increment mode
mov     edx, [ebp].IOControl
mov     eax, CNTL_AUTO_INC
out     dx, ax
; Set Adapter Ram Address
mov     edx, [ebp].IOMsgRamPtr
xor     eax, eax
out     dx, ax
; to zero

mov     edx, [ebp].IOMsgRamPtr
xor     eax, eax
out     dx, ax
; Set Adapter Ram Address
mov     edx, [ebp].IOMsgRamPtr
xor     eax, eax
out     dx, ax
; to zero

mov     ecx, MipsCodeSize
mov     edx, [ebp].IOMsgRam
lea     esi, MipsCode
cld
CopyToAdapterLoop:
lodsw
out     dx, ax
; Get Mips Word
; Send it to adapter

```

```
dec     ecx
jnz     CopyToAdapterLoop
```

```
; Verify the download by reading the data back
```

```
mov     edx, [ebp].IOControl
mov     eax, CNTL_AUTO_INC
out     dx, ax
```

```
; Set to auto-increment mode
```

```
mov     edx, [ebp].IOMsgRamPtr
xor     eax, eax
out     dx, ax
```

```
; Set Adapter Ram Address
; to zero
```

```
mov     ecx, MipsCodeSize
mov     edx, [ebp].IOMsgRam
lea     esi, MipsCode
cld
```

```
; Number of words to copy
; adapter ram port
; esi -> mips code
```

```
VerifyAdapterLoop:
```

```
xor     eax, eax
```

```
; Get Mips Word
```

```
lodsw
```

```
; Send it to adapter
```

```
mov     edi, eax
in     ax, dx
cmp     edi, eax
lea     eax, MsgBadRam
jne     DriverInitErrorReturn
dec
```

```
dec     ecx
jnz     VerifyAdapterLoop
```

```
; Register our interrupt handler with the OS.
```

```
mov     edx, [ebp].IOStatus
in     ax, dx
```

```
; Clear old errors
```

```
; Set RBD base address
```

```
mov     cx, [ebp].IORbBufLen
mov     eax, RBD_BASE_ADDR
out     dx, ax
mov     edx, [ebp].IOMsgRamPtr
out     dx, ax
```

```
mov     edx, [ebp].IORbNum
mov     eax, ADAP_RBD_NUM
out     dx, ax
mov     ecx, eax
```

```
mov     edx, [ebp].IORbBufLen
mov     eax, RBD_BUFFER_SIZE
out     dx, ax
```

```
mov     edx, [ebp].IOControl
mov     eax, CNTL_AUTO_INC
out     dx, ax
```

```
mov     edx, [ebp].IOMsgRam
```

```
SetupBuffersLoop:
```

```
xor     eax, eax
out     dx, ax
```

```
mov     eax, RBD_BUFFER_SIZE
out     dx, ax
```

```
dec     ecx
jnz     SetupBuffersLoop
```

```
; Enable the adapter.
```

```
movzx   ecx, [ebx].MLIDInterrupt
mov     esi, CNTL_IRQ3
```

```
mov     edx, 8
cmp     ecx, 8h
je       EnabledPC
mov     esi, CNTL_IRQ4
```

```
mov     ecx, 10h
cmp     ecx, 4
je       EnabledPC
mov     esi, CNTL_IRQ5
```

```
mov     edx, 20h
cmp     ecx, 5
je       EnabledPC
mov     esi, CNTL_IRQ9
```

```
mov     edx, 2h
cmp     ecx, 8
je       EnabledPC
mov     esi, CNTL_IRQ10
```

```
mov     ecx, 4h
cmp     ecx, 10
je       EnabledPC
mov     esi, CNTL_IRQ11
```

```
mov     edx, 8h
cmp     ecx, 11
je       EnabledPC
mov     esi, CNTL_IRQ12
```

```
mov     ecx, 10h
cmp     ecx, 12
je       EnabledPC
mov     esi, CNTL_IRQ15
```

```
mov     edx, 80h
mov     esi, CNTL_IRQ15
```

```
EnabledPC:
mov     [ebp].PicMask, edx
not     edx
```

```
mov     [ebp].PicUnMask, edx
mov     [ebp].PicAddress, 21h
cmp     [ebx].MLIDInterrupt, 8
```

```
jb       ClearOurInterrupt
mov     [ebp].PicAddress, 0a1h
ClearOurInterrupt:
```

```
mov     edx, [ebp].PicAddress
in     al, dx
and     eax, [ebp].PicUnMask
out     dx, al
```

```
mov     eax, esi
mov     edx, [ebp].IOControl
or     eax, CNTL_RX_EN OR CNTL_CPU_EN OR CNTL_INT_EN OR CNTL_SINGL_INT_0
R CNTL_AUTO_INC OR CNTL_SOURCEPUT
out     dx, ax
mov     [ebp].IOEnableValue, eax
```

```
cmp     DebugMask, 0
je       OpenScreenExit
push     4e524353h
```

```
lea     eax, ScreenResourceName
push     eax
push     DriverModuleHandle
call     AllocateResourceTag
lea     esp, [esp + (3 * 4)]
or     eax, eax
```

```

;
je      OpenscreenExit
mov     ScreenTag, eax

push    offset DPCScreen
push    eax
push    offset ScreenName
call    Openscreen
lea     esp, [esp + (3 * 4)]
or      eax, eax
jne     OpenscreenExit

push    offset NLName
push    0
push    DriverModuleHandle
call    GetNLNames
lea     esp, [esp + (3 * 4)]

push    0
push    offset Day
push    offset Month
push    offset Year
push    0
push    offset MinorVer
push    offset MajorVer
push    offset NLName
push    offset DebugScreenHeader
push    DPCScreen
push    OutputScreen
call    esp, [esp + (8 * 4)]
lea     esp, [esp + (8 * 4)]

push    Year
push    Day
push    Month
push    MinorVer
push    MajorVer
push    offset NLName
push    offset DebugScreenHeader
push    DPCScreen
push    OutputScreen
call    esp, [esp + (8 * 4)]
lea     esp, [esp + (8 * 4)]

OpenscreenExit:
;.....
; Initialize RxControl and Filter entries.
;.....
lea     edi, [ebp].RxControl
mov     ecx, MAX_CHAN
xor     eax, eax
mov     edx, RBD_NOT_USED

InitRxControlLoop:
mov     [edi].RxChannel, edx
mov     [edi].RxESR, eax
add     edi, size RX_CNTL
dec     ecx
jne     InitRxControlLoop
lea     edi, [ebp].Filter
mov     ecx, MAX_ADDR

InitFilterLoop:
mov     [edi].FilterChannel, edx
mov     [edi].FilterTotalCount, eax
mov     [edi].FilterSeqCount, eax
mov     [edi].FilterSeqNum, eax

```

```

; Location of Handle storage
; Screen Resource Tag
; Name of screen object
;.....
add     edi, size FilterStruct
dec     ecx
jne     InitFilterLoop

; Test Interrupts.
;.....
; Set ISR to test routine.
;
mov     [ebp].GotInterrupt, 0 ; Clear test flag.
mov     ecx, [ebp].ISRTag
push    0 ; ECX = ISR resource tag.
push    0 ; No ExtraIOFlag offset.
push    0 ; ShareFlag.
push    0 ; End of chain flag.
push    0 ; ISR Resource tag.
lea     ecx, TestDriverISR
push    eax
movzx   eax, [ebp].MLIDInterrupt
push    eax
call    SetHardwareInterrupt
lea     esp, [esp + (6 * 4)]
or      eax, eax
lea     eax, BadISRMsg
jnz     DriverInitErrorReturn ; EAX -> Error Message.
; Exit if so.

mov     edx, [ebp].IOControl
mov     eax, [ebp].IOEnableValue
or      eax, CNTL_FORCEINT
out     dx, ax

sti
mov     eax, [ebp].TimerTag
push    eax
push    18
call    DelayMyself
add     esp, (2 * 4)
cli

movzx   eax, [ebp].MLIDInterrupt
lea     edx, TestDriverISR
push    edx
push    eax
call    ClearHardwareInterrupt
lea     esp, [esp + (2 * 4)]

mov     eax, [ebp].IOEnableValue
out     dx, ax

lea     eax, NoInterruptMsg
cmp     [ebp].GotInterrupt, 0 ; Did we get it?
je      DriverInitErrorReturn ; Jump if not.

; Error message.
; Pass ISR Entry point.
; Pass Interrupt #.
; Give interrupt back.
; Clean up stack.
;.....
; EBX -> Frame Data Space(Config Table).
; EBP -> Adapter Data Space.
; Let MSM Set Hardware Interrupt.
;.....
call    MSMSetHardwareInterrupt

```

```

; Jump if error.
jnz DriverInitError
;.....\
; Set TxFreeCount to make TSM happy.
;.....\
mov [ebp].MSMTxFreeCount, 32 ; Allow 32 transmits sim
ultaneously.

mov eax, 1 ; Schedule call back in 18 ticks
call MSMScheduleIntrTimerCallBack
jnz DriverInitError ; Jump if error.
call DriverReset ; Initialize NIC.
jnz DriverInitErrorReturn ; Exit if error resetting.
mov [ebp].FirstTimeInit, 0 ; Disable DriverReset from
; testing the hardware again.

dec [ebp].AdapterResetCount ; Adjust reset count.
call MSMRegisterMLID ; Register MLID.
jnz DriverInitError ; Jump if error.

; Lets see if the adapter is locked up.
mov eax, [ebp].TimerTag
push 18
call DelayMyself
add esp, (2 * 4)

call RefreshMipsStats
test [ebp].MipsRxEnables, 8000000h ; This shouldn't be big
lea eax, LockedAdapterMsg
jne DriverInitErrorReturn

cmp DebugMask, 0
je DriverInitExit
movzx eax, [ebp].MLIDInterrupt
push eax
movzx eax, [ebp].MLIDIOPortsAndLengths
push eax
push offset DebugInitOK
push DPCScreen
push OutputScreen
lea esp, [esp + (4 * 4)]

DriverInitExit:
mov [ebp].MLIDMaxRecvSize, 1400
xor eax, eax
CPOP
ret

DriverInitErrorReturn:
push eax ; Save error message.
call MSMReturnDriverResources ; Return resources.
mov eax, DPCScreen
or eax, eax
je DriverInitErrorScreenClosed
push eax
call CloseScreen
lea esp, [esp + (1 * 4)]

;.....\
DriverInit:
subttl -- DriverShutdown --
page
;.....\
BEGIN_MANUAL_ENTRY ( DriverShutdown, DPC/API/SHUTDOWN )
;.....\
Name: DriverShutdown
Description: This routine will turn off the NIC.

On Entry:
EAX N/A
EBX @ Frame Data Space
ECX 0 if Permanent Shutdown
EDX N/A
EBP @ Adapter Data Space
ESI N/A
EDI N/A

Note: Interrupts are disabled.

On Return:
EAX 0 if successful
EBX Preserved
ECX Preserved
EDX Destroyed
EBP Preserved
ESI Preserved
EDI Preserved

Flags:
Note: Interrupts preserved.

Remarks:
This routine is called by the MSM media module.
It is called at process time.

See Also:
ETHERTSN\ETHERTSMShutdown

END_MANUAL_ENTRY
;.....\
DriverShutdown proc
or ecx, ecx
jne DriverShutdownAdapter
mov eax, [ebp].AgentRemoveRoutine
or eax, eax
je DriverShutdownAdapter
call eax

```

mov [ebp].AgentRemoveRoutine, 0

DriverShutdownAdapter:

pushfd

cld

mov edx, [ebp].IOControl

xor eax, eax

out dx, ax

mov edx, [ebp].IOStatus

in ax, dx

in

or ecx, ecx

jne DriverShutdownExit

mov edx, [ebp].IOControl

mov eax, CNTRL_MRESET

out dx, ax

out

mov eax, DPCScreen

or eax, eax

je DriverShutdownScreenClosed

push eax

call CloseScreen

lea esp, [esp + (1 * 4)]

mov DPCScreen, 0

DriverShutdownScreenClosed:

mov eax, [ebp].ProtocolBindID

or eax, eax

je DriverShutdownExit

push eax

call UnregisterEventNotification

add esp, (1 * 4)

mov eax, [ebp].ProtocolUnbindID

or eax, eax

je DriverShutdownExit

push eax

call UnregisterEventNotification

add esp, (1 * 4)

add

DriverShutdownExit:

popfd

xor eax, eax

ret

DriverShutdown endp

subttl -- DriverRemove --

page

.....\

;; BEGIN_MANUAL_ENTRY(DriverRemove, DPC/API/REMOVE)

;; Name: DriverRemove

;;

;; Description: This routine call the MSM to return our resources.

;;

;; On Entry:

;;

;;

;;

;;

;;

;;

;;

EDI N/A

Note: Interrupts are in any state.

;

;

;

;

;

;

;

;

;

;

;

;

;

;

;

;

;

;

;

;

;

;

;

;

;

;

;

;

;

;

;

;

;

;

;

;

;

;

;

;

;

;

;

;

;

;

;

;

;

;

;

;

;

;

;

;

;

;

;


```

/* Interface between Helius DPCNE and Hughes DPCPE */

extern "C" {
#include <semaph.h>
#include "sys_win.h"
#include "dpcutils.h"
#include "dbsinwin.h"
#include "VIRTUAL"
#include "docagent.h"
#include <assert.h>

int PD_ESR(ECB*);
void DIOHangup(void);
void DPCPDTerminate(void);
void DPCPDBackground(void);
void DPCFileMain(void* arg); // thread
}

#include "sfwatch.h"
#include "sfqview.h"
#include "sfqpar.h"

unsigned long GetTickCount(void) {
    return clock() * 1000 / CLOCKS_PER_SEC;
}

extern int DIOState;
extern LONG DIOXmitCount;
extern LONG DIOMaxBufferSize;
extern LONG DIOrcvCount;
extern LONG DIOConn;

int DIOGetCurrentState(void) {
    if 1
        return (DIOState == DIOS_CONN && DIOConn == DIO_CONN_PACKAGE) ? DIOS_CONN : DL
        OS_IDLE;
    #else
        return DIOState;
    #endif
}

int DIOPortEmpty(void) {
    #if 1
        return DIOXmitCount == 0;
    #else
        return DIOAndCommEmpty();
    #endif
}

int DIOPortOpen(void) {
    return AIOPortHandle != (-1);
}

int DIOGetStatus(DIOStatus* pStatus) {
    if (pStatus == 0)
        return (-1);
    if (!DIOPortOpen())
        return (-1);
    pStatus->iState = DIOGetCurrentState();
    pStatus->iXmitBytesBuffered = DIOXmitCount;
    pStatus->iXmitBufferSize = DIOMaxBufferSize;
    pStatus->iRcvBytesBuffered = DIOrcvCount;
    pStatus->iRcvBufferSize = DIOBUFSIZE;
    return 0;
}

int DIOGetBufferSize(void) {
    DWORD DIOExtendInactivityTimer(long) {
    }
    void DIOHangup(void) {
    }
    void DIODispatch(void) {
    }

    /* Returns whether the adapter can gain access to the passed group ID
     * The group ID includes a version number.
     */
    long DLAPI CDBCheckGroupID(CdbCfg_t *cfg)
    {
        if (!find_pacau(cfg->groupid, cfg->ver) != NULL)
            return(CAS_IMPLICIT);
        if (!find_dacau(cfg->groupid, cfg->ver) != NULL)
            return(CAS_AUTHENTICATED);
        if (!find_eacau(cfg->groupid, cfg->ver) != NULL)
            return(CAS_EXPLICIT);
        return(CAS_ERROR);
    }

    /* Returns a version number which increments when there have been
     * ANY changes to the adapter's conditional access.
     */
    long DLAPI CDBCheckCACHange(void)
    {
        return CDBVersion;
    }

    struct PID {
        PID() { DPCFilePID = GetThreadID(); }
        ~PID() { DPCFilePID = 0; }
    };

    struct Semaphore {
        LONG handle;
        Semaphore(long initial = 0) { handle = OpenLocalSemaphore(initial); }
        ~Semaphore() { if (handle) CloseLocalSemaphore(handle); }
        void Signal(void) { SignalLocalSemaphore(handle); }
        LONG Wait(int milliseconds = -1) { return TimedWaitOnLocalSemaphore(handle,
            (LONG)milliseconds); }
        LONG value(void) { return ExamineLocalSemaphore(handle); }
        LONG operator --(void);
    };

    inline LONG Semaphore::operator --(void) {
        LONG v = value();
        if (v == 0)
            return v;
        WaitOnLocalSemaphore(handle);
        return v - 1;
    }

    Semaphore* DPCPDSemaphore;
    SfxDispatcher* pDispatcher;
    QUEUVIEWER* pQueueViewer;
    ECBQueue DPCPDQueue;

```

```

int PD_ESR(ECB* ecb) {
    Enqueue_IntsDisabled(&DPCPDQueue, ecb);
    return 0;
}

long Bicddsigntext(char* p_string,
                   unsigned long size,
                   char* p_sign) {
    return DIOSigntext(p_string, size, p_sign);
}

long BicddgetSN(char* p_serial_num) {
    DIOSetsN(p_serial_num);
    return 0;
}

long Bicddopenchannel(BICDD_CHANNEL_CONFIG* channel_config) {
    if (channel_config->num_addresses != 1)
        return 1;

    DPCPDSemaphore = new Semaphore();
    if (DPCPDSemaphore->handle == 0) {
        delete DPCPDSemaphore;
        DPCPDSemaphore = 0;
        return 2;
    }
    DPCPDQueue.semaphore = DPCPDSemaphore->handle;

    // there is actually an overflow here IRT channel being a short!
    long ret = DIOOpenChannel(channel_config->address(0),
                              PD_ESR,
                              (LONG*) &channel_config->channel);

    if (ret) {
        delete DPCPDSemaphore;
        DPCPDSemaphore = 0;
        DPCPDQueue.semaphore = 0;
        return ret;
    }

    long Bicddclosechannel(unsigned long channel) {
        long ret = DIOCloseChannel(channel);
        if (ret)
            return ret;
        delete DPCPDSemaphore;
        DPCPDSemaphore = 0;
        DPCPDQueue.semaphore = 0;
        while (DPCPDQueue.head) {
            ECB* ecb = Dequeue(&DPCPDQueue);
            CLSReturnRCvECB(ecb);
        }
        return 0;
    }

    .....
    *
    *      ELEMENTS SECTION
    *      ( Elements Table support )
    *
    * .....
}

CDBelement_t Elements(MAXELEMENTS);

static find_element_by_mac(MACAddr_t mac)
{
    int k;
    int ret = -1;

    for(k = 0; k < MAXELEMENTS; k++) {
        if(Elements[k].in_use == 'Y' &&
            memcmp(&Elements[k].e_mac, &mac, sizeof(mac)) == 0) {
            ret = k;
            break;
        }
    }
    return ret;
}

static add_element(unsigned long channel, ID id, unsigned char ver,
                  MACAddr_t mac, char pack_feed)
{
    int k, ret = CAS_OK;

    if(find_element_by_mac(mac) != -1)
        return(CAS_DUPLICATE_ADDR);
    for(k = 0; k < MAXELEMENTS; k++)
        if(Elements[k].in_use != 'Y')
            break;
    if(k == MAXELEMENTS)
        ret = CAS_ERROR;
    else {
        Elements[k].channel = channel;
        memcpy(&Elements[k].e_id, &id, sizeof(id));
        memcpy(&Elements[k].e_mac, &mac, sizeof(mac));
        Elements[k].in_use = 'Y';
        Elements[k].packfeed = pack_feed;
    }
    return ret;
}

static find_element_id(ID id, unsigned char ver)
{
    int k;
    int ret = -1;

    for(k = 0; k < MAXELEMENTS; k++) {
        if(Elements[k].in_use == 'Y' &&
            memcmp(&Elements[k].e_id, &id, sizeof(id)) == 0 &&
            Elements[k].e_ver == ver) {
            ret = k;
            break;
        }
    }
    return ret;
}

static del_element_by_mac(MACAddr_t mac)
{
    int k, ret = CAS_ERROR;

    for(k = 0; k < MAXELEMENTS; k++) {
        if(Elements[k].in_use == 'Y' &&
            memcmp(&Elements[k].e_mac, &mac, sizeof(mac)) == 0) {
            ret = CAS_OK;
            Elements[k].in_use = 'N';
            break;
        }
    }
    return ret;
}

```



```

static char* PREGKEY_Reconcile = "Reconcile";
static char* PREGKEY_EnableDebug = "EnableDebug";
static char DBS_NAME[] = "_FILE_";
static const char magic_key[] = {
    0x11, 0x11, 0x11, 0x11, 0x11, 0x11, 0x11, 0x11, 0x11, 0x11
};

/*****

EXPORTED FUNCTION

DPCCancelDownload(LONG fileID)

Description:
    This routine cancels the download of the file associated
    with the fileID if one is pending. This means closing
    any open file handles and stopping the modem thread.

Input:
    fileID          - File ID of file to cancel

Output:
    nothing

Returns:
    0                if download was canceled

*****/

static struct {
    LONG control;
    LONG ret;
    LONG fileID;
    BOOL cancel;
} crossover;

LONG DPCCancelDownload(LONG fileID)
{
    while (crossover.control)
        delay(100);
    crossover.fileID = fileID;
    crossover.cancel = TRUE;
    crossover.control = GetThreadId();
    while (crossover.control)
        delay(100);

    /* Force the help package status to idle */
    UpdateHelpPortal();
    return crossover.ret;
}

LONG DPCDownloadAFfile(LONG fileID)
{
    while (crossover.control)
        delay(100);
    crossover.fileID = fileID;
    crossover.cancel = FALSE;
    crossover.control = GetThreadId();
    while (crossover.control)
        delay(100);
    return crossover.ret;
}

void DPCPDterminate(void) {
    PDispatcher->Terminate();
}

void DPCPDBackground(void) {
    PDI_FillList_cross();

    if (crossover.control) {
        LONG fileID = crossover.fileID;
        if (fileID != fsm.getFileID() && fsm.unique(fileID) != SFX_OK)
            crossover.ret = (LONG)(-1);
        else if (crossover.cancel) {
            crossover.ret = fsm.dispatch(fileID, SFXFSM_FILE_NOT_WANTED);
            PDispatcher->CancelLoadingFileID(fileID);
        }
        else if (fsm.isRequestable(fileID)) {
            fsm.dispatch(fileID, SFXFSM_PRECOMMIT);
            crossover.ret =
                fsm.dispatch(fileID,
                    fsm.isForSale(fileID) ? SFXFSM_PURCHASE : SFXFSM_FILE_WRITE);
        }
        D);
        sendret:
        ResumeThread(crossover.control);
        crossover.control = 0;
    }

    // this is adapted from sfxdemlp.cpp WinMain and dpcfile.c DPCFileMain
    void DPCFileMain(void* arg) {
        PID pid;
        if (!PackageDelivery)
            return;
        DbsProcInit("DPCPD");
        if ((arg && strcmp((char*)arg, "rebuild") == SUCCESS) ||
            DPCGetProfileInt(PROF_PACKAGEDELIVERY, PREGKEY_RebuildonStartup, 0)) {
            DPCSetProfileInt(PROF_PACKAGEDELIVERY, PREGKEY_RebuildonStartup, 0);
            int n = fsm.Rebuild();
            if (n >= 0) {
                DBS_SEND_TRACEI(0, "File database rebuilt with %d entries restored",
                    n);
            }
            else {
                DBS_SEND_TRACEI("File database rebuild failed");
            }
            return;
        }
        // wait until DIOBoard initialized
        while (DIOBoard == 0) {
            if (ExitingFlag)
                return;
            delay(500);
        }
        PDispatcherLo = new SfxDispatcherLo();
        if (!PDispatcherLo) {
            DBS_SEND_ERROR(DBS_FATAL, "Could not construct SfxDispatcherLo");
            goto cleanup;
        }
    }
}

```

```
)
    pDispatcher = new SfxDispatcher();
    if (!pDispatcher) {
        DBS_SEND_ERROR(DBS_FATAL, "Could not construct SfxDispatcher");
        goto cleanup;
    }
    pQueueViewer = new QUEUEVIEWER(PDI_Updatedisplay);
    if (!pQueueViewer) {
        DBS_SEND_ERROR(DBS_FATAL, "Could not construct QUEUEVIEWER");
        goto cleanup;
    }

    if (!DPGetProfileInt(PROF_PACKAGEDELIVERY, PREGKEY_Reconcile, 1))
        fsm.ReconcileWith(frd);
    cdb.rebuildDB();

    while (!ExitingFlag) {
        pDispatcher->Run();
        DPCPBackground();
    }

    pDispatcher->Stop(5000);

cleanup:
    delete pQueueViewer;
    delete pDispatcher;
    delete pDispatcherLo;
}
```

```

#include "dpcagent.h" /* Our header file */

/* *fix* conflicting types */
#define AllocateResourceTag __AllocateResourceTag__
#include <advanced.h>
#include <Allocateresource.h>

#define millclock() (GetHighResolutionTimer() / 10)

#define activityTimer ECB_DriverWorkspace.DWS_132val

/* various flags that control the filter */
#define FILTER_DATA_ON_RST
#define WIDEN_TCP_WINDOW
#define TCP_ACK_LATENCY /* 10 */
/* if used at all, define "only" 1 of the following */
#define DPCinetMaxQueuedBytes /* 4096 */
#define DPCinetMaxQueuedBytes 64
/* if defined(DPCinetMaxQueuedBytes) && defined(DPCinetMaxQueuedPackets)
 * error Only 1 of DPCinetMaxQueuedBytes and DPCinetMaxQueuedPackets allowed
 */
#endif

/* various flags that control the tunnel */
#define TUNNEL_ONLY_TCP

#define IP_VERS(x) ((BYTE*)(x)[0] >> 4)
#define IP_HD_LEN(x) ((BYTE*)(x)[0] & 0x0f)
#define IP_TOS(x) ((BYTE*)(x)[1])
#define IP_TOT_LEN(x) ((WORD*)(x)[1])
#define IP_FLAG_FRAG(x) ((WORD*)(x)[3])
#define IP_PROTO(x) ((BYTE*)(x)[9])
#define IP_SUM(x) ((WORD*)(x)[5])
#define IP_SRC_ADDR(x) ((LONG*)(x)[3])
#define IP_DST_ADDR(x) ((LONG*)(x)[4])

#define IPPROTO_IPENCAP 0x04

#define UDP_SRC_PORT(x) ((WORD*)(x)[0])
#define UDP_DST_PORT(x) ((WORD*)(x)[1])

#define TCP_SRC_PORT(x) ((WORD*)(x)[0])
#define TCP_DST_PORT(x) ((WORD*)(x)[1])
#define TCP_ACKNUM(x) ((LONG*)(x)[2])
#define TCP_CODE(x) ((BYTE*)(x)[13])
#define TCP_WINDOW(x) ((WORD*)(x)[7])
#define TCP_CSUM(x) ((WORD*)(x)[8])

#define TCP_FIN 0x01
#define TCP_SYN 0x02
#define TCP_RST 0x04
#define TCP_PSH 0x08
#define TCP_ACK 0x10
#define TCP_URG 0x20

FCBQueue TxQ;
ECBQueue NewQ;

struct ResourceTagStructure* TxChainRTag = 0;
struct ResourceTagStructure* TxRCNRTag = 0;
LONG TxChainID;
struct ResourceTagStructure* RxChainRTag = 0;
struct ResourceTagStructure* RxECBRTag = 0;
LONG RxChainID;
LONG DPC_IP_Address = 0;
#define BYTE ConnectionMark(65536 / 8);

#endif

```

```

/* *fix* conflicting types */
#define inline inline
#endif /* __GNUC__ */

/* ECB Manipulation */
static inline void ReleaseECB(ECB* ecb) {
    if (DIOSStats) {
        #ifdef DPCinetMaxQueuedBytes
            if (DIOSStats->QDepth == ecb->ECB_DataLength) < 0)
                DIOSStats->QDepth = 0;
        #endif
        #ifdef DPCinetMaxQueuedPackets
            --DIOSStats->QDepth;
        #endif
    }
    --TxECBRTag->RTResourceCount;
    CLSIFastSendComplete(ecb);
    #ifdef LOG_ECB_ACTIVITY
        FastLogMsg(LogECBHandle, (LogClientHandle, LogECBHandle, TRUE,
            "TINET Release(%08lx)\n", ecb));
    #endif /* LOG_ECB_ACTIVITY */
}

inline void Enqueue_IntsDisabled(ECBQueue* q, ECB* ecb) {
    ecb->ECB_NextLink = 0;
    if (q->tail)
        q->tail->ECB_NextLink = ecb;
    ecb->ECB_PreviousLink = q->tail;
    q->tail = ecb;
    if (q->head == 0)
        q->head = ecb;
    SignalLocalSemaphore(q->semaphore);
}

void Enqueue(ECBQueue* q, ECB* ecb) {
    _disable();
    Enqueue_IntsDisabled(q, ecb);
    _enable();
}

ECB* Dequeue(ECBQueue* q) {
    ECB* ecb;
    _disable();
    ecb = q->head;
    if (ecb == 0) {
        _enable();
        return 0;
    }
    q->head = ecb->ECB_NextLink;
    if (q->head == 0)
        q->tail = 0;
    else
        q->head->ECB_PreviousLink = 0;
    _enable();
    ecb->ECB_NextLink = ecb->ECB_PreviousLink = 0;
    return ecb;
}

```

```

void Remove(ECBQueue* q, ECB* ecb) {
    _disable();
    if (ecb->ECB_NextLink)
        ecb->ECB_NextLink->ECB_PreviousLink = ecb->ECB_PreviousLink;
    else
        q->tail = ecb->ECB_PreviousLink;
    if (ecb->ECB_PreviousLink)
        ecb->ECB_PreviousLink->ECB_NextLink = ecb->ECB_NextLink;
    else
        q->head = ecb->ECB_NextLink;
    _enable();
    ecb->ECB_NextLink = ecb->ECB_PreviousLink = 0;
}

LONG InetQueuePacket(ECB* ecb, LONG board, void* chainID) {
    board = board;
    chainID = chainID;
    /* not used */
    /* not used */

    /* only handle IP packets */
    if (! (LONG*)ecb->ECB_ProtocolID != 0 ||
        *(WORD*)(ecb->ECB_ProtocolID + 4) != htons(0x0800))
        return 1;

    #ifdef LOG_ECB_ACTIVITY
    if (LogECBHandle) {
        int TGID = SetThreadGroupID(DPC_TGID);
        LogMsg(LogClientHandle, LogECBHandle, FALSE,
            "INET Enqueue(%08lx)\n", ecb);
        SetThreadGroupID(TGID);
    }
    #endif /* LOG_ECB_ACTIVITY */
    Enqueue(&newq, ecb);
    return 0;
}

LONG InetControl(void) {
    return 0x1ffff1;
}

void ClearConnection(WORD port) {
    if (Scanbits(ConnectionMask, port, port+2) == port) {
        BitClear(ConnectionMask, port);
        --DIOStats->TXOKMultipleCollisions;
    }
}

int AllocateConnection(WORD port) {
    if (Scanbits(ConnectionMask, port, port+2) != port) {
        /* see if there is a connection left */
        if (DIOStats->TXOKMultipleCollisions < DPCMaxConnections) {
            /* allocate the new connection */
            BitSet(ConnectionMask, port);
            ++DIOStats->TXOKMultipleCollisions;
            return 1;
        }
        return 0;
    }
    return 1;
}

LONG ConnectionLimiter(ECB* ecb, LONG board, void* chainID) {
    BYTE* IPHeader = ecb->ECB_Fragment[0].FragmentAddress;
    BYTE* TCPHeader = 0;
    board = board;
    chainID = chainID;
    /* not used */
    /* not used */

    /* only handle IP packets */
    if (! (LONG*)ecb->ECB_ProtocolID != 0 ||
        *(WORD*)(ecb->ECB_ProtocolID + 4) != htons(0x0800))
        return 1;

    /* double check stats, hopefully upper layer is kosher, but */
    if (DIOStats == 0) {
        releaseECB:
        --RXECBRTag->RTResourceCount;
        CLSIFastSendComplete(ecb);
        return 0;
    }

    /* only check TCP packets to our interface */
    if (IP_PROTO(IPHeader) != IP_PROTO_TCP ||
        IP_DST_ADDR(IPHeader) != DPC_IP_Address)
        return 1;

    TCPHeader = IPHeader + IP_HD_LEN(IPHeader) * 4;
    if (ecb->ECB_Fragment[0].FragmentLength < (TCPHeader + 20) - IPHeader))
        return 1;

    if (TCP_CODE(TCPHeader) & (TCP_FIN|TCP_RST)) {
        /* release the connection */
        ClearConnection(ntohs(TCP_DST_PORT(TCPHeader)));
    }
    else if (TCP_CODE(TCPHeader) & TCP_SYN) {
        /* allocate the connection */
        if (!AllocateConnection(ntohs(TCP_DST_PORT(TCPHeader))))
            goto releaseECB;
    }
    return 1;
}

/* IP Manipulation */
#ifdef 0
char *chksum (BYTE *buf, unsigned cnt)
{
    static unsigned char crc_bytes[2];
    BYTE rbl;
    WORD rax, rcx;
    int redx;
    BYTE *rdsi;

    crc_bytes[0] = crc_bytes[1] = 0;

    rcx = cnt;
    rdsi = buf;
    rbl = rcx;
    rcx = rcx >> 1;
    redx = 0;
    if (rcx != 0)
    {
        while (rcx--)
        {
            rax = *((WORD *)rdsi);
            rdsi += 2;
        }
    }
}

```

```

if (redx & 0xffff0000)
    redx++;
    redx &= 0x0000ffff;
    redx += rax;
}
if (redx &= 0x0000ffff)
    redx &= 0x0000ffff;
    redx++;
}
if (rbl & 1)
{
    rax = 0;
    rax = *rdsi;
    redx += rax;
    if (redx &= 0x0000ffff)
        redx++;
}
    redx = ~redx;
    crc_bytes[0] = redx & 0xff;
    crc_bytes[1] = (redx >> 8) & 0xff;
    return (char *)crc_bytes;
};

#endif

#ifdef __GNUC__
/*
 * This is a version of ip_compute_csum() optimized for IP headers, which
 * always checksum on 4 octet boundaries.
 * This version is constructed from various places in the linux and Hughes
 * sources.
 */

static inline unsigned short ip_fold_lcomp_csum(unsigned long sum) {
    unsigned short csum;
    __asm__ ("movl %w1, %w0\n\t"
            "shrl $16, %1\n\t"
            "addw %w1, %w0\n\t"
            "adcl $0, %w0\n\t"
            "notw %w0"
            : "=a" (csum)
            : "b" (sum));
    return csum;
}

static inline unsigned short ip_fast_csum(unsigned short * buff, int wlen) {
    unsigned long sum = 0;
    if (wlen) {
        unsigned long eax;
        /* Suggested speedup:
         * 1:
         *   movl (%esi), %ebx
         *   lea (%esi+4), %esi
         *   adcl %ebx, %eax
         *   decl %ecx
         *   jnz 1b
         *   adcl $0, %eax
         *   movl %eax, %ebx
         *   shrl $16, %eax
         *   addw %ebx, %eax
         *   adcl $0, %eax
         */
    }
}

xorl $0xffff, %eax
*/
__asm__ ("clic\n"
        "1:\n"
        "lodsl\n\t"
        "adcl %3, %0\n\t"
        "loop 1b\n\t"
        "adcl $0, %0\n\t"
        : "=r" (sum), "=S" (buff), "=c" (wlen), "=a" (eax)
        : "0" (sum), "1" (buff), "2" (wlen));
}
return ip_fold_lcomp_csum(sum);
}

#define chksum(b, l) ip_fast_csum(b, (l) / 4)

static inline unsigned short ip_adjust_csum(unsigned short oldcsum,
        unsigned short oldval,
        unsigned short newval) {
    unsigned long sum = ((unsigned short)~oldcsum);
    sum += ((unsigned short)~oldval);
    sum += newval;
    return ip_fold_lcomp_csum(sum);
}

#endif /* __GNUC__ */

static int DummyFrame(FRAG_DESC* frag) { /* not used */
    frag = frag;
    return 0;
}

int (*DPCDDropFrame)(FRAG_DESC* frag) = DummyFrame;

void FilterQueue(void* arg) {
    ECB* ecb;
    ECB* rover;
    BYTE* IP;
    BYTE* TCP;
    int excess;
    arg = arg; /* not used */

    RenameThread(GetThreadId(), "DPCAgent Filter");
    for (;;) {
        if (ExitingFlag)
            return;
        TmMediaiConLocalSemaphore(NewQ.semaphore, 1000);
        if (!NewQ.head)
            continue;
        ecb = Dequeue(&NewQ);
        ecb->activityTimer = millisecond();
        IP = ecb->ECB_Fragment[0].FragmentAddress;
        if (DIOSStats == 0) {
            releaseECB:
            DPCDropFrame((FRAG_DESC*)&ecb->ECB_FragmentCount);
            --TxECBRTag->RTResourceCount;
            CLSIFastSendComplete(ecb);
            #ifdef LOG_ECB_ACTIVITY
            FastLogMsg(LOGECBHandle, (LogClientHandle, LOGECBHandle, TRUE,
                    "TINET Release(%08lx)\n", ecb));
            #endif
        }
    }
}

```



```

#endif /* LOG_ECB_ACTIVITY */
continue;
}

/* always send a fragmented or routed packet */
if (ecb->ECB_Fragment[0].FragmentLength < 20 ||
    IP_FLAG_FRAGMENT(IP) & htons(0x3fff) ||
    IP_SRC_ADDR(IP) != DPC_IP_Address) {
    enqueueTxQ:
    #ifndef DPCnetMaxQueuedBytes
    if (DIOStats->QDepth > DPCnetMaxQueuedBytes) {
        ++DIOStats->RetryTxCount;
        goto releaseECB;
    }
    #endif
    DIOStats->QDepth += ecb->ECB_DataLength;
    #ifndef DPCnetMaxQueuedPackets
    if (DIOStats->QDepth > DPCnetMaxQueuedPackets) {
        ++DIOStats->RetryTxCount;
        goto releaseECB;
    }
    #endif
    ++DIOStats->QDepth;
    Enqueue(&TxQ, ecb);
    continue;
}

excess = ecb->ECB_Fragment[0].FragmentLength - IP_HD_LEN(IP) * 4;
if (excess > 0) {
    TCP = IP + IP_HD_LEN(IP) * 4;
}
else {
    TCP = ecb->ECB_Fragment[1].FragmentAddress + (-excess);
    excess += ecb->ECB_Fragment[1].FragmentLength;
}

if (IP_PROTO(IP) != IPPROTO_UDP)
    goto filterUDP;

if (IP_PROTO(IP) != IPPROTO_TCP)
    goto enqueueTxQ;

filterTCP:
if (excess < 20)
    goto enqueueTxQ;

if (TCP_CODE(TCP) & TCP_SYN) {
    if (!allocateConnection(&txos(TCP_SRC_PORT(TCP))))
        goto releaseECB;
    /* scan for duplicate in TxQ */
    for (rover = TxQ.head; rover = rover->ECB_NextLink) {
        BYTE roverIP = rover->ECB_Fragment[0].FragmentAddress;
        BYTE* roverTCP;
        excess = (rover->ECB_Fragment[0].FragmentLength -
            IP_HD_LEN(roverIP) * 4);
        if (excess > 0) {
            roverTCP = roverIP + IP_HD_LEN(roverIP) * 4;
        }
        else {
            roverTCP = rover->ECB_Fragment[1].FragmentAddress + (-excess);
            excess += rover->ECB_Fragment[1].FragmentLength;
        }
        if (rover->ECB_Fragment[0].FragmentLength >= 20 &&
            excess >= 20 &&
            IP_FLAG_FRAGMENT(roverIP) & htons(0x3fff) == 0 &&
            IP_PROTO(roverIP) == IPPROTO_TCP &&
            TCP_CODE(roverTCP) == TCP_CODE(TCP)) {
                /* scan for redundancy in TxQ */
                for (rover = TxQ.head; rover = rover->ECB_NextLink) {
                    if (TCP_CODE(TCP) & (TCP_RST|TCP_FIN)) {
                        ClearConnection(&txos(TCP_SRC_PORT(TCP)));
                        goto enqueueTxQ;
                    }
                    if (TCP_CODE(TCP) & TCP_ACK) {
                        #ifndef WIDEN_TCP_WINDOW
                        WORD oldwin = TCP_WINDOW(TCP);
                        WORD newwin = ntohs(oldwin);
                        if (newwin < 40000) {
                            newwin += (newwin >> 1);
                            newwin = htons(newwin);
                            TCP_WINDOW(TCP) = newwin;
                            TCP_CSUM(TCP) = ip_adjust_csum(TCP_CSUM(TCP),
                                oldwin,
                                newwin);
                        }
                        #endif
                    }
                }
            }
        }
    }
}

#endif /* WIDEN_TCP_WINDOW */
if (TCP_CODE(TCP) & (TCP_URG|TCP_PSH))
    goto enqueueTxQ;
#endif TCP_ACK_LATENCY
ecb->activityTimer = millisecond() + TCP_ACK_LATENCY;
#endif

```

```

BYTE* roverIP = rover->ECB_Fragment[0].FragmentAddress;
BYTE* roverTCP;
excess = (rover->ECB_Fragment[0].FragmentLength -
IP_HD_LEN(roverIP) * 4);
if (excess > 0) {
    roverTCP = roverIP + IP_HD_LEN(roverIP) * 4;
}
else {
    roverTCP = rover->ECB_Fragment[1].FragmentAddress + (-excess);
    excess += rover->ECB_Fragment[1].FragmentLength;
}
if (rover->ECB_Fragment[0].FragmentLength >= 20 &&
    excess >= 20 &&
    (IP_FLAG_FRAG(roverIP) & htons(0x3fff)) == 0 &&
    IP_PROTO(roverIP) == IPPROTO_TCP &&
    IP_DST_ADDR(roverIP) == IP_DST_ADDR(IP) &&
    TCP_DST_PORT(roverTCP) == TCP_DST_PORT(TCP) &&
    IP_SRC_ADDR(roverIP) == IP_SRC_ADDR(IP) &&
    TCP_SRC_PORT(roverTCP) == TCP_SRC_PORT(TCP) &&
    TCP_CODE(roverTCP) & TCP_ACK &&
    htonl(TCP_ACKNUM(roverTCP)) + htonl(TCP_WINDOW(roverTCP)) <
    htonl(TCP_ACKNUM(TCP)) + htonl(TCP_WINDOW(TCP))) {
    /* move ACK information over to TxQ and release this packet */
    TCP_CSUM(roverTCP) = ip_adjust_csum(TCP_CSUM(roverTCP),
TCP_WINDOW(roverTCP),
TCP_WINDOW(TCP));
TCP_CSUM(roverTCP) = ip_adjust_csum(TCP_CSUM(roverTCP),
(WORD)TCP_ACKNUM(roverTCP),
(WORD)TCP_ACKNUM(TCP));
TCP_CSUM(roverTCP) = ip_adjust_csum(TCP_CSUM(roverTCP),
TCP_ACKNUM(roverTCP)>>16,
TCP_ACKNUM(TCP)>>16);
TCP_CSUM(roverTCP) = ip_adjust_csum(TCP_CSUM(roverTCP),
(WORD)TCP_ACKNUM(roverTCP),
(WORD)TCP_ACKNUM(TCP));
TCP_CSUM(roverTCP) = ip_adjust_csum(TCP_CSUM(roverTCP),
TCP_ACKNUM(roverTCP)>>16,
TCP_ACKNUM(TCP)>>16);
TCP_ACKNUM(roverTCP) = TCP_ACKNUM(TCP);
TCP_WINDOW(roverTCP) = TCP_WINDOW(TCP);
++DIOStats->TxAbortExcessCollisions;
goto releaseECB;
}
}
goto enqueueTxQ;
}
goto enqueueTxQ;
}
filterUDP:
{
    BYTE* UDP = TCP;
    BYTE* DNS;
    /* ECB contents determined by inspection, there are safer methods */
    if (excess < 8)
        goto enqueueTxQ;
    /* filter DNS only */
    if (UDP_DST_PORT(UDP) != htons(53))
        goto enqueueTxQ;
    excess -= 8;
    DNS = (excess > 0) ? (UDP + 8) : ECB->ECB_Fragment[1].FragmentAddress;
    for (rover = TxQ.head; rover; rover = rover->ECB_NextLink) {
        BYTE* roverIP = rover->ECB_Fragment[0].FragmentAddress;
        BYTE* roverUDP;
        excess = (rover->ECB_Fragment[0].FragmentLength -
IP_HD_LEN(roverIP) * 4);
        if (excess > 0) {
            roverUDP = roverIP + IP_HD_LEN(roverIP) * 4;
        }
        else {
            roverUDP = rover->ECB_Fragment[1].FragmentAddress + (-excess);
            excess += rover->ECB_Fragment[1].FragmentLength;
        }
        if (rover->ECB_Fragment[0].FragmentLength >= 20 &&
            excess >= 8 &&
            (IP_FLAG_FRAG(roverIP) & htons(0x3fff)) == 0 &&
            IP_PROTO(roverIP) == IPPROTO_UDP &&
            IP_DST_ADDR(roverIP) == IP_DST_ADDR(IP) &&
            UDP_DST_PORT(roverUDP) == UDP_DST_PORT(UDP) &&
            IP_SRC_ADDR(roverIP) == IP_SRC_ADDR(IP) &&
            UDP_SRC_PORT(roverUDP) == UDP_SRC_PORT(UDP) &&
            (roverDNS = ((excess == 8) > 0) ?
(roverUDP + 8) :
(rover->ECB_Fragment[1].FragmentAddress)) &&
            * (LONG*)DNS == * (LONG*)roverDNS) {
            ++DIOStats->TxAbortLateCollision;
            goto releaseECB;
        }
        * (LONG*)DNS == * (LONG*)roverDNS) &&
        ++DIOStats->TxAbortLateCollision;
        goto releaseECB;
    }
    goto enqueueTxQ;
}
}
}
/* SLIP, PPP, Modem Manipulation */
#define MAX_READ_BUF 128
int InetState = MODEM_IDLE;
static BYTE Slipendpkt[1] = (END);
int WaitingLines = 0, NextWait = 0;
char WaitingBuffer[MAX_READ_BUF];
int WaitingIndex = 0;
LONG ConnectingTimeout = 0;
LONG ConnectingRedial = FALSE;
int BaudRate[] =
{
    2400, /* 0 */
    3600, /* 1 */
    4800, /* 2 */
    7200, /* 3 */
    9600, /* 4 */
    19200, /* 5 */
    38400, /* 6 */
    57600, /* 7 */
    115200 /* 8 */
};
void InitLogin()
{
    int i;
    char *nextWait;
    WaitingLines = 0;
    if (DIOCfg.auto_login)
    {
        WaitingIndex = 0;
        WaitingBuffer[WaitingIndex] = '\0';
        NextWait = 0;
    }
}

```



```

0, 0, /* ident */
0, 0, /* fragment */
0x7f, /* ttl */
4, /* IP in IP (encapsulation) */
);
#define IPHeaderIdent (*(WORD*)&IPHeader[4])
int IPSendRoutine(ECB *ecb)
{
    FRAG_DESC *fragStruc = alloca(sizeof(LONG) * (sizeof(FRAGMENTSTRUCT) * (
ecb->ECB_FragmentCount + 2)));
    WORD frame_size = ecb->ECB_DataLength;
    int options_collapsed = 1;
    LONG curFrag = 0;
    BYTE ecbIPHeader = ecb->ECB_Fragment[0].FragmentAddress;

    /* initialize the copy of the tcb fragStruc */
    memcpy(fragStruc,
ecb->ECB_FragmentCount,
sizeof(LONG) * (sizeof(FRAGMENTSTRUCT) * ecb->ECB_FragmentCount));

    if (frame_size < DIOCfg.mtu &&
#ifdef TUNNEL_ONLY_TCP
/* UDP doesn't need tunnel header */
(ecbIPHeader[9] != IPPROTO_TCP) ||
#endif
/* either do "routed" packets */
((LONG*)&ecbIPHeader[12] != DPC_IP_Address))
    goto skipFragger;

    memset(&fragStruc->FragmentDesc[fragStruc->FragmentCount],
0,
sizeof(FRAGMENTSTRUCT) * 2);
    frame_size += IP_TUNNEL_SIZE;

    /* fill IPHeader with tunnel data, including IP/gateway addresses,
    * and prepend to frag list.
    */
    *(WORD*)&IPHeader[2] = htons(frame_size);
    ++IPHeaderIdent;
    *(WORD*)&IPHeader[10] = 0; /* checksum, for now */
    *(LONG*)&IPHeader[12] = DIOCfg.ip_address;
    *(LONG*)&IPHeader[16] = DIOCfg.gateway_address;
    memmove(&fragStruc->FragmentDesc + 1,
fragStruc->FragmentDesc,
sizeof(FRAGMENTSTRUCT) * fragStruc->FragmentCount);
    fragStruc->FragmentDesc[0].FragmentAddress = IPHeader;
    fragStruc->FragmentDesc[0].FragmentLength = IP_TUNNEL_SIZE;
    ++fragStruc->FragmentCount;
    ++curFrag;
    *(WORD*)&IPHeader[10] = checksum(WORD *)IPHeader,
IP_TUNNEL_SIZE);

    while (frame_size > DIOCfg.mtu)
    {
        /*
        * Shucks. Have to fragment the packet.
        * This algorithm is roughly per RFC791.
        */
        LONG OIHL = fragStruc->FragmentDesc[0].FragmentLength;
        BYTE OMF = IPHeader[6] & 0x20;
        LONG NPH = (DIOCfg.mtu - OIHL) & 0xffff;
        WORD TL = OIHL + NPH;

        IPHeader[6] |= 0x20; /* set More Fragments */
        *(WORD*)&IPHeader[2] = htons(TL);
        *(WORD*)&IPHeader[10] = 0; /* clear checksum */
        *(WORD*)&IPHeader[10] = checksum(WORD *)IPHeader, OIHL);

        /*
        * Now fake out the fragStruc to reflect TL.
        * Hang on to enough information to remove the TL less OIHL
        * later.
        */
        TL -= OIHL;
        frame_size -= TL;
        while (TL > 0 &&
fragStruc->FragmentDesc[curFrag].FragmentLength <= TL)
        {
            TL -= fragStruc->FragmentDesc[curFrag].FragmentLength;
            ++curFrag;
        }
        if (TL > 0)
        {
            /*
            * This frag gets split into 2 pieces.
            */
            memmove(&fragStruc->FragmentDesc + curFrag + 1,
fragStruc->FragmentDesc + curFrag,
sizeof(FRAGMENTSTRUCT) *
(fragStruc->FragmentCount - curFrag));
            ++fragStruc->FragmentCount;
            fragStruc->FragmentDesc[curFrag].FragmentLength = TL;
            ++curFrag;
            fragStruc->FragmentDesc[curFrag].FragmentLength -= TL;
            fragStruc->FragmentDesc[curFrag].FragmentAddress = ((ch
ar*)fragStruc->FragmentDesc[curFrag].FragmentAddress) + TL;
        }
        TL = fragStruc->FragmentCount - curFrag + 1;
        fragStruc->FragmentCount = curFrag;

        if (DPCTxFrame(fragStruc) == 0)
            return 0;

        if (!options_collapsed) {
            LONG offset = 20;
            while (offset < OIHL && IPHeader[offset]) {
                if (IPHeader[offset] & 0x80) /* copy */
                    offset += IPHeader[offset + 1];
                else /* collapse */
                    LONG len = IPHeader[offset + 1];
                    memcopy(IPHeader + offset + len,
OIHL - (offset + len));
                    OIHL -= len;
            }
            offset = fragStruc->FragmentDesc[0].FragmentLength;
            fragStruc->FragmentDesc[0].FragmentLength = (OIHL + 3) &
0x3c;
            memset(IPHeader + OIHL,
0,
fragStruc->FragmentDesc[0].FragmentLength - OIHL);
            IPHeader[0] = 0x40 | (fragStruc->FragmentDesc[0].Fragmen
frame_size -= offset - fragStruc->FragmentDesc[0].Fragme
options_collapsed = 1;
        }
    }
}

```

```

/*
 * Adjust the frag list to "remove" the fragment just sent.
 */
memmove(fragStruc->FragmentDesc + 1,
        fragStruc->FragmentDesc + curFrag,
        sizeof(FRAGMENTSTRUCT) * TL);
fragStruc->FragmentCount = TL;
curFrag = 1;

/* compute new IPHeader values: fragment offset */
*(WORD*)(IPHeader[6]) = htonl(ntohs(*(WORD*)(IPHeader[6])) &
                                0xffff
                                + (NFB / 8));

IPHeader[6] |= 0MF;
if (frame_size <= DIOCfg.mtu)
{
    *(WORD*)(IPHeader[2]) = htons(frame_size);
    *(WORD*)(IPHeader[10]) = 0; /* Clear checksum */
    *(WORD*)(IPHeader[10]) = checksum(WORD *)IPHeader,
        fragStruc->FragmentDesc
[0].FragmentLength);
    break;
}
}

skipFragger:
/* send the remaining (possibly ALL) portion of the frame */
if (DIOCfg.frame(fragStruc) == 0)
    return 0;

if (DIOStats)
{
    ++DIOStats->TotalTxPacketsCount;
    if ((DIOStats->TotalTxOKByteCountLow += ecb->ECB_DataLength) <
        ++DIOStats->TotalTxOKByteCountHigh; /* wrapped */)
    {
        return 1;
    }
}

static void EmptyESR(ECB * ecb) {
}

unsigned char RawEnvelope[14] = {
    0x00, 0x00, 0x0C, 0x0A, 0x0B, 0x0C,
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
    0x08, 0x00,
};

ECB RawECB = {
    0, /* ECB_NextLink */
    0, /* ECB_PreviousLink */
    0, /* ECB_Status */
    EmptyESR, /* ECB_ESR */
    -1, /* ECB_StackID */
    0, /* ECB_ProtocolID */
    0, /* ECB_BoardNumber */
    0, /* ECB_ImmediateAddress */
    (0), /* ECB_DriverWorkspace */
    (0), /* ECB_ProtocolWorkspace */
    -1, /* ECB_DataLength */
    -1, /* ECB_FragmentCount */
    (RawEnvelope, sizeof(RawEnvelope))
};

FRAGMENTSTRUCT RawFrag[15] = {
    IPHeader, sizeof(IPHeader),
};

#define RawSendRoutineDebug RawSendRoutineOpt
int RawSendRoutineOpt(FRAG_DESC * fragStruc) {
    int i;
    for (i = 20000; --i > 0; ) {
        if (RawECB.ECB_Status == 0)
            goto rawSend;
        if (ExitingFlag)
            return 0;
        ThreadSwitchWithDelay();
    }
    return 0;
rawSend:
    i = fragStruc->FragmentCount;
    RawECB.ECB_FragmentCount = i + 1;
    memcpy(RawFrag,
        fragStruc->FragmentDesc,
        i * sizeof(FRAGMENTSTRUCT));
    RawECB.ECB_DataLength = sizeof(RawEnvelope);
    while (i > 0)
        RawECB.ECB_DataLength += RawFrag[--i].FragmentLength;
    CLSndPacket(&RawECB);
    return 1;
}

void DisplayWaitStatus(void)
{
    char statusStr[80];
    char *waitingStr;

    if (WaitingLines == 0)
        return;

    waitingStr = (char *)&DIOCfg.wait_for_1[NextWait*30];
    MPrintf(statusStr, MSG("Modem Status: Waiting for \"%s\\n\", wait
tingStr);
    UpdateModemStr(statusStr);
}

void InetStateChange(int state) {
    if (DIOCfg.out_protocol == OUT_NETWORK) {
        InetState = PROTOCOL_CONNECTED;
        return;
    }
    switch (state) {
        case DLOS_IDLE:
        case DLOS_DISC_1:
        case DLOS_DISC_2:
        case DLOS_DISC_3:
        case DLOS_DISC_4:
            InetState = MODEM_IDLE;
            if (ConnectingRedial) {
                DIOEndConn();
                ConnectingRedial = FALSE;
            }
            break;
        case DLOS_CONN:
            InetState = MODEM_CONNECTED;
            InitLogIn();
            if (InetAsleep)
                ResumeThread(DPC_InetPID);
            break;
    }
}

```

```

default:
    break;
}

/*.....
 * FUNCTION: Convert Internet address Address
 *
 * DESCRIPTION: converts a character string containing the Internet address
 * into a form that BIG DD understands.
 * e.g. 139.85.124.06 (BB.55.7C.06) into 067C558B0000
 *.....*/
void convert_address(char *lpszIPaddress)
{
    char *p;
    int i = 0;
    char tmp[20], tmp1[10];
    tmp[0] = 0;
    while((p=strchr(lpszIPaddress, '.')) != NULL)
    {
        i = atoi(p+1);
        sprintf(tmp1, "%02X", 477), i);
        strcat(tmp, tmp1);
        *p = 0;
    }
    i = atoi(lpszIPaddress);
    sprintf(tmp1, "%02X", 478), i);
    strcat(tmp, tmp1);
    strcpy(tmp, MSG("0000", 479));
    strcpy(lpszIPaddress, tmp);
}

MACAddr_t      HIAddr;
LONG            InetChannel;

void make_hi_key(chunk *key)
{
    int i;
    LONG sn;
    BYTE serialNum[9];
    BYTE serialNumPacked[3];
    BYTE x;

    DIOGetSN(serialNum);
    sn = atoi(serialNum);
    sprintf(serialNum, MSG("%06lx", 480), sn);

    pack_mac_addr(serialNumPacked, 3, serialNum, 6);
    x = serialNumPacked[0];
    serialNumPacked[0] = serialNumPacked[2];
    serialNumPacked[2] = x;

    key->b[0] = serialNumPacked[0] ^ 0xff;
    key->b[1] = serialNumPacked[1] ^ 0xff;
    key->b[2] = serialNumPacked[2] ^ 0xff;
    for(i = 3; i < 8; i++)
        key->b[i] = 0x00 ^ 0xff;

    MACBuildAddr(serialNum, MAC_HI, 0, &HIAddr);
}

void InetChangeProtocol(void)
{
    switch (DIOCfg.out_protocol) {
        case OUT_PPP:
            DPCtxFrame = DebugFlag ? PPPSendRoutineDebug : PPPSendRoutineOpt;
            break;
        case OUT_NETWORK:
            void (*ControlEntryPoint)(void) = 0;
            struct DriverConfigurationStructure *dvrcfg = 0;
            if (CLSLGetMLIDControlEntry(DIOCfg.net_interface,
                                         &ControlEntryPoint))
            {
                goto skipDriver;
            }
            dvrcfg = (struct DriverConfigurationStructure *)
                CommandMID(DIOCfg.net_interface, 0, (LONG)ControlEntryP
                memcpy(RawEnvelope + 6, dvrcfg->DNodeaddress, 6);

            skipDriver:
            RawECB.ECB_BoardNumber = DIOCfg.net_interface;
            memcpy(RawEnvelope, DIOCfg.net_addr, 6);
            DPCtxFrame = DebugFlag ? RawSendRoutineDebug : RawSendRoutineOpt;
            break;
        case OUT_SLIP:
            DPCtxFrame = DebugFlag ? SLIPSendRoutineDebug : SLIPSendRoutineOpt;
            break;
    }
    InetStateChange(DIOS_DISC_4);
    DIOEndConn();
}

int ProcessLogin(void)
{
    BYTE value;
    char *sendStr, *waitStr;
    char sendBuf[40];
    LONG nextTimeout;

    /* No use trying if we aren't even connected */
    /* Get out if we're done */
    if (WaitingLines == 0 || DIOCfg.auto_login == FALSE)
    {
        return(TRUE);
    }
    if (!DIOConnected())
        return(FALSE);

    /* Timeout if we've waited too long for this wait */
    if (ConnectingTimeout == 0)
    {
        ConnectingTimeout = GetCurrentTime() + DIOCfg.wait_timeout * 1
    }
    if (GetCurrentTime() > ConnectingTimeout)

```

```

(
    if (ConnectingRedial == FALSE)
    {
        /* First timeout. Send return and try again. */
        ConnectingRedial = TRUE;
        InitLogin();
        DIOsend(MSG("\r", 181), 1, DLO_INET_TIMEOUT);
        return(FALSE);
    }
    DIOendConn();
    return(FALSE);
}

DisplayWaitStatus();

while (DIOReceive(&value, 1) != 0)
{
    if (DebugFlag)
        putchar(value);
    if (value != '\r' && value != '\n')
    {
        WaitingBuffer[WaitingIndex++] = value;
        WaitingBuffer[WaitingIndex] = 0;
        if (WaitingIndex > (MAX_READ_BUF-1))
            WaitingIndex = 0;
    }
    waitStr = (char *)&DIOCfg.wait_for[1][NextWait * 30];
    if (!strcmp(WaitingBuffer, waitStr) != NULL)
    {
        sendStr = (char *)&DIOCfg.send_1[NextWait * 30];
        NMSprintf(sendBuf, MSG("%s\r", 558), sendStr);
        DIOsend(sendBuf, CStrlen(sendBuf), DLO_INET_TIMEOUT);
        NextWait++;
        WaitingIndex = 0;
        WaitingBuffer[WaitingIndex] = '\0';
        WaitingLines--;
        if (WaitingLines == 0)
        {
            DIOupdateModemStr();
            return(TRUE);
        }
        DisplayWaitStatus();
        nextTimeout = DIOCfg.wait_timeout_1 + NextWait;
        ConnectingTimeout = GetCurrentTime() +
            ((nextTimeout) ? (nextTimeout * 18) : (5
* 18));
        return(FALSE);
    }
    else if (value == '\r')
    {
        WaitingIndex = 0;
        WaitingBuffer[WaitingIndex] = '\0';
    }
}

return(FALSE);

int ConnectProtocol(void)
{
    int ccode;
    if (DIOCfg.out_protocol == OUT_SLIP)
        delay(1000); /* time to "settle" */
    return 1;
}

void TinetProtocolBind(LONG parameter)
{
    struct EventProtocolBindStruct epbs =
        (struct EventProtocolBindStruct) parameter;
    if (epbs->boardNumber == DIOBoard &&
        epbs->protocolNumber == 1/*PROTOCOL_ID_TCP/IP*/)
    {
        extern LONG DPCNextRegistrationCheck;
        DPCGetIPaddress(&DPC_IP_Address);
        DPCNextRegistrationCheck = 0;
    }
}

/*.....*/
InetMain(void *parm)
{
    Description:
        Main thread for Turbo Internet handling.

    Input:
        parm

    Output:
        nothing

    Returns:
        nothing

    .....*/

void InetMain(void *parm)
{
    time_t nextStartConn = 0;
    LONG removedCount = (LONG)(-1);
    long millisecondDelay = 0;
    LONG protocolBindHandle =
        RegisterForEvent(EVENT_PROTOCOL_BIND, TinetProtocolBind, 0);
    parm = parm; /* unused */

    NewO_semaphore = OpenLocalSemaphore(0);
    TxQ_semaphore = OpenLocalSemaphore(0);
    BeginThread(FilterQueue, 0, 0, 0);

    TxChainRTag = AllocateResourceTag(NLMHandle,
        MSG("Turbo Inet TxPrescan Chain", 476)
        LSLTxPrescanStackSignature);
    TxECBRTag = AllocateResourceTag(NLMHandle,
        MSG("Turbo Inet Transmit Packets", 619),
        ECBSignature);
    RxChainRTag = AllocateResourceTag(NLMHandle,
        "Turbo Inet RxPrescan Chain",
        LSLRxPrescanStackSignature);
    RxECBRTag = AllocateResourceTag(NLMHandle,
        MSG("Turbo Inet Receive Packets", 619),

```

```

DPCGetIPAddress(&DPC_IP_Address);
ECBSignature);

if (DIOCfg.out_protocol == OUT_NETWORK)
    InetState = PROTOCOL_CONNECTED;

mainloop:
while (!ExitingFlag)
{
    InetAsleep = TRUE;
    if (millidelay > 55)
        delay(millidelay);
    else if (millidelay > 0)
        ThreadSwitchWithDelay("LowPriority")();
    else
        ThreadSwitch();
    InetAsleep = FALSE;

    while (DIOBoard && removedCount != DIORemovedCount)
    {
        BYTE address(8);
        BYTE szBicBCDAddress(20);
        struct DriverStatsStructure* stats = 0;
        LONG ip_address = htonl(DIOCfg.ip_address);

        removedCount = DIORemovedCount;
        /* Enable internet reception */

        /* Yuk. We'll change this later to get rid these extra s
        NMSprintf(szBicBCDAddress, MSG("%d.%d.%d.%d", 620),
            (ip_address >> 24) & 0xff,
            (ip_address >> 16) & 0xff,
            (ip_address >> 8) & 0xff,
            (ip_address) & 0xff);

        convert_address(szBicBCDAddress);
        if (!pack_mac_addr(address, 6,
            szBicBCDAddress, CSzLen(szBicBCDAddr
            ess)))
        {
            /* UpdateModemStr(MSG("ERROR: could not pack mac
            address\n", xxxx)); */
            millidelay = 500;
            break;
        }

        /* Sending an esr address of -1 tells MLID to handle rec
        option */
        if (DIOOpenChannel(address,
            (int (*)())0xffffffff,
            &InetChannel))
        {
            millidelay = 500;
            removedCount = (LONG)(-1);
            break;
        }
        if (ExitingFlag)
            break;
        DIOAddHIAddr(InetChannel, (BYTE *)&HIAddr);
        DPCGetMLIDStats(&stats);
        DIOStats->TxOKMultiPleCollisions = 0;
        if (CULRegisterPrescanTxChain(TxChainInRTag,
            DIOBoard,
            3, /* next to last */
            &TxChainID,
            ConnectionLimiter,
            InetControl,
            RxECBRtag))
        {
            millidelay = 500;
            removedCount = (LONG)(-1);
            break;
        }

        if (CULRegisterPrescanRxChain(RxChainInRTag,
            DIOBoard,
            3, /* next to last */
            &RxChainID,
            ConnectionLimiter,
            InetControl,
            RxECBRtag))
        {
            millidelay = 500;
            removedCount = (LONG)(-1);
            break;
        }

        if (DIOCfg.out_protocol == OUT_PPP &&
            InetState == PROTOCOL_CONNECTED)
        {
            PPPBackground();
        }

        if (TxQ.head == 0 &&
            (InetState <= MODEM_CONNECTING ||
            InetState >= PROTOCOL_CONNECTED))
        {
            TimeWaitOnLocalSemaphore(TxQ.semaphore, 200);
            millidelay = 0;
            continue;
        }

        switch (InetState)
        {
            case MODEM_CONNECTED:
                if (!ProcessLogin())
                {
                    millidelay = 500;
                    break;
                }
                InetState = LOGIN_CONNECTED;
                /* fallthru */
            case LOGIN_CONNECTED:
                if (!ConnectProtocol())
                {
                    DIOEndConn();
                    millidelay = 15 * 1000;
                    break;
                }
                InetState = PROTOCOL_CONNECTED;
                millidelay = 1;
                break;
            case PROTOCOL_CONNECTED:
                LONG count = 0;
                if (DIOCfg.out_protocol != OUT_NETWORK &&
                    AIOWriteStatus(AIOportHandle, &count, 0))
                {
                    millidelay = 200;
                    break;
                }
        }
    }
}

```



```

if (count == 0)
{
    LONG millisecond = millisecond();
    ECB* ecb;
    ecb = TxQ.head;
    millisecond = 100;
    while (ecb->activityTimer > millisecond)
    {
        LONG diff = ecb->activityTimer - millisecond;

        if (diff < millisecond)
            millisecond = diff;
        if ((ecb = ecb->ECB_NextLink) == 0)
            goto mainloop;
    }
    Remove(&TxQ, ecb);
    if (ecb->activityTimer < millisecond() - 60000) {
        if (ecb->activityTimer)
            +DIOStats->TxAbortExDeferral;
    }
    else
        IPendRoutine(ecb);
    ReleaseECB(ecb);
    if (DIOCfg.out_protocol != OUT_NETWORK)
        AIOWriteStatus(AIOPortHandle, &count, 0);
}
millisecond = (count *
    10 * /* Tx bits with framing */
    1000 / /* milliseconds */
    BaudRate[DIOCfg.tinet_baud_index]);
break;
}
case MODEM_IDLE:
    if (nextStartConn < time(0))
    {
        InitLogin();
        DIOStartConn(DIO_INET_TIMEOUT);
        InetState = MODEM_CONNECTING;
        nextStartConn = time(0) + 30;
        /* fallback */
    }
    default:
        millisecond = 10 * 1000;
        break;
    }
}
}

DIOCloseChannel(InetChannel);

CLSIDeregisterPrescanRxChain(RxChainID);
CLSIDeregisterPrescanTxChain(TxChainID);
while (TxQ.head)
    ReleaseECB(Degueue(&TxQ));
while (NewQ.head)
    ReleaseECB(Degueue(&NewQ));
CloseLocalSemaphore(TxQ.semaphore); TxQ.semaphore = 0;
CloseLocalSemaphore(NewQ.semaphore); NewQ.semaphore = 0;

UnregisterForEvent(protocolBindHandle);

DPCinetPID = 0;
return;
}

```